# Sur une méthode de routage des messages dans les architectures parallèles à mémoire distribuée : application à la grille torique

Mustapha Boukhalfa Hadim

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT ETIENNE

N° d'ordre : 158 ID

**THESE**
présentée

pour obtenir le grade de **DOCTEUR EN SCIENCE** de

**L'UNIVERSITE JEAN MONNET et de L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT ETIENNE**

Spécialité : **Informatique**

PAR

**Mustapha Boukhalfa HADIM**

# Sur une Méthode de Routage des Messages dans les Architectures Parallèles à Mémoire Distribuée : Application à la Grille Torique

Soutenue le 30 Juin 1997 devant la Commission d'examen composée de

| | | |
|---|---|---|
| Mr. | Bernard PEROCHE | Président |
| Mr. | Joffroy BEAUQUIER | Rapporteur |
| Mr. | Yousef SAAD | Rapporteur (excusé) |
| Mme. | Marie-Claude HEYDEMANN | Examinateur |
| Mr. | Jean-Claude KÖNIG | Examinateur |
| Mr. | Jean AZEMA | Examinateur |
| Mr. | Jean-Jacques GIRARDOT | Examinateur |
| Mr. | Ibrahima SAKHO | Directeur |

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT ETIENNE

N° d'ordre: 158 ID

**THESE**
présentée

pour obtenir le grade de **DOCTEUR EN SCIENCE** de

**L'UNIVERSITE JEAN MONNET et de L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT ETIENNE**

Spécialité: **Informatique**

PAR

**Mustapha Boukhalfa HADIM**

# Sur une Méthode de Routage des Messages dans les Architectures Parallèles à Mémoire Distribuée: Application à la Grille Torique

Soutenue le 30 Juin 1997 devant la Commission d'examen composée de

| | | |
|---|---|---|
| Mr. | Bernard PEROCHE | Président |
| Mr. | Joffroy BEAUQUIER | Rapporteur |
| Mr. | Yousef SAAD | Rapporteur (excusé) |
| Mme. | Marie-Claude HEYDEMANN | Examinateur |
| Mr. | Jean-Claude KÖNIG | Examinateur |
| Mr. | Jean AZEMA | Examinateur |
| Mr. | Jean-Jacques GIRARDOT | Examinateur |
| Mr. | Ibrahima SAKHO | Directeur |

# Résumé

Dans les architectures parallèles à mémoire distribuée, la communication entre processus est un des facteurs de performance les plus importants pour les applications. Le système qui en a la charge, i.e, *le noyau de communication*, doit intégrer une fonctionnalité essentielle pour de telles architectures : *le routage des messages.* Cette fonctionnalité est assurée par une composante spécifique du noyau de communication : *le noyau de routage*, dont le rôle est l'acheminement d'un message d'un nœud émetteur vers un nœud récepteur.

L'acheminement des messages nécessite une *stratégie de routage* qui spécifie les chemins de communication pour toute paire de processeurs (source,destination) du réseau d'interconnexion. Une telle stratégie de routage doit satisfaire d'une part, des *critères de correction* et d'autres part, des *critères d'efficacité.*

Le but de cette thèse est la conception de stratégies de routage pour les réseaux de processeurs qui satisfont à la fois, les critères de correction et les critères d'efficacité. Nous proposons une méthode de conception de stratégies de routage, permettant par une démarche incrémentale, de satisfaire les deux types de critère : *la communication multi-niveaux* et le *schéma de communication primaire* associé.

Pour mesurer l'efficacité de la méthode, nous l'appliquons à un réseau particulier : *la grille torique.* Les différents algorithmes de routage obtenus sont corrects et très efficaces.

Nous proposons également une technique d'implantation de notre méthode de routage, permettant le calcul des tables de routage directement sur le réseau de processeurs. Cette technique permet ainsi l'obtention d'un système *auto-constructif.*

**Mots clés** : Architectures parallèles, Communication, Routage des messages, Communication multi-niveaux, Schéma de communication primaire, Grille torique.

# Abstract

In distributed memory parallel architectures, interprocess communication is one of the main efficiency factor for the applications. The system which is in charge of this task, i.e, *the communication kernel*, must integrate an essential functionality for such architectures : *message routing*. This functionality is ensured by a specific component of the communication kernel : the *routing kernel*, whose purpose is to convey a message from a sender node to a receiver node.

Messages conveying requires a *routing strategy* which specify the communication paths for each pair of (source,destination) processors of the interconnection network. Such routing strategy must satisfy in one hand, correction criteria and in other hand, *efficiency criteria*.

The purpose of this thesis is the design of routing strategies for networks of processors, which satisfy at the same time, correction criteria and efficiency criteria. We propose a method to design routing strategies, which allows by an incremental approach, to satisfy both of the criteria : the *multi-level communication* and the associated *primary communication scheme*.

To prove the efficiency of the method, we apply it to a particular topology : The *torus* network. The several obtained routing algorithms are correct and very efficient.

We propose also a technique to implement our routing method, which allows to compute the routing tables directly on the target network processors. Thus, this technique allows to obtain a *self constructing* system.

**Keywords** : Parallel architectures, Communication, Message routing, Multi-level communication, Primary communication scheme, Torus network.

# Remerciements

*J'exprime mes plus vifs remerciements à tous les membres du jury:*

*Monsieur* BERNARD PEROCHE, *Professeur à l'Ecole Nationale Supérieure des Mines de Saint Etienne, pour m'avoir accueilli dans l'ex DIA et pour l'honneur qu'il me fait de présider mon jury de thèse,*

*Messieurs* JOFFROY BEAUQUIER, *Professeur à l'Université de Paris XI centre d'Orsay et* YOUSEF SAAD, *Professeur à l'Université du Minnesota au USA, pour avoir accepté, avec une grande gentillesse, la lourde tâche d'évaluer mon travail, malgré leurs nombreuses contraintes internationales,*

*Madame* MARIE-CLAUDE HEYDEMANN, *Professeur à l'Université de Paris XI centre d'Orsay, pour l'honneur qu'elle me fait de participer au jury,*

*Monsieur* JEAN-CLAUDE KÖNIG, *Professeur à l'université d'Evry, pour avoir bien voulu être examinateur, sans aucune réserve,*

*Monsieur* JEAN AZEMA, *Maitre de Conférence à l'université* JEAN MONNET, *pour avoir accepté, avec une grande gentillesse, de participer au jury,*

*Monsieur* JEAN-JACQUES GIRARDOT, *Maitre de Recherches à l'Ecole des Mines de Saint Etienne, pour avoir accepter d'être mon directeur de thèse,*

*Monsieur* IBRAHIMA SAKHO, *Ingénieur de Recherches Habilité à l'Ecole des Mines de Saint Etienne, pour avoir dirigé mes recherches et pour m'avoir tant appris.*

*Je tiens à remercier chaleuresement, Madame* ANNIE CORBEL BOURGEAT *et Monsieur* MICHEL BEIGBEDER, *pour m'avoir fait confiance au tout début.*

*Je remercie Monsieur* ABDELMADJID BOUABDALLAH, *pour le papier que nous avons écrit ensemble. Je remercie également, Monsieur* ROLAND JEGOU, *pour ses conseils éclairés.*

*Mes remerciements vont également à l'encontre des membres passés et présents du centre SIMADE, permanents et thésards:*

*Madame* HÉLÈNE SAYET *pour sa gentillesse, Mesdames* MARIE-LINE BARNÉOUD, LILLIANE BROUILLET *et* ZAHIA MAZER *pour leurs disponibilités,* JEAN-PIERRE TAVERNIER, FLORENCE DUJARDIN, MICHEL CHATARD, DOMINIQUE MICHELUCCI, JEAN MICHEL MOREAU, JORI LEONARDON, CLAUDETTE SAYETTAT, ANTOINE PAUZE, SAMI AIT AOUDIA, MOHAND OURABAH BENOUAMER, BERTRAND JULLIEN, NADIA KABACHI, GRACIELA ROMANO, PHILIPPE BISCONDI, HAKIM KAHLOUCHE, JEAN-LUC MAILLOT, GILLES MATHIEU, FRANÇOISE SAUZET, PIERRE BASSOMO, BICH-LIEN DOAN, MAHDI HANNOUN, KONRAD SZAFNICKI, REDOUANE SENOUNE, SOPHIE PERREAR *et* XAVIER BAY.

*Je remercie le personnel du service de reprographie pour avoir assurer la reproduction de ce mémoire.*

*Je tiens à remercier chaleuresement Monsieur* CHOUKRI-BEY BENYELLES. *Je remercie mes amis,* SALIM LEMDANI, KARIM ABED-MERAIEM, SALIM AGOUDJIL, NABIL BEDJAOUI, SAID BABACI, YACINE ATIF, HAKIM BENKEDDAD, MOUNIR HAHHAD, ABDERAHIM SEDQUI, *et* SOUAD BERRAMI.

*Je remercie mes cousins,* HAMID, RABAH, MADJID, ABDELLAH *et* OUERDIA *pour leurs conseils précieux. J'exprime mes plus vifs remerciements à Mr.* ABDELKRIM MOULAI.

*Je n'oublie pas mes parents, mes frères et sœurs et ma femme* SORAYA, *sans qui tous cela n'aurait pu être fait. Je les remercie pour leur affection et la patience dont ils ont fait preuve durant mes années de thèse.*

# Table des matières

# Chapitre 1

# Introduction générale

## 1.1 Du séquentiel au parallèle

De nos jours, le **parallélisme** est unanimement admis comme la *solution* aux besoins en puissance de calcul de plus en plus grands des applications. Il vient compléter les solutions initiales qui consistaient en l'amélioration des algorithmes et l'augmentation de la puissance des composants électroniques.

En effet, la première approche poursuivie pour augmenter la puissance d'un ordinateur a été d'améliorer la puissance de ses composants élémentaires, ainsi que leur technologie de construction et leur organisation dans cet ordinateur, tout en conservant le modèle de fonctionnement associé à VON NEUMANN [61].

La figure 1.1 illustre l'architecture logique d'un tel calculateur. Le programme utilisateur ainsi que ses données sont stockés au niveau d'une *mémoire centrale.* A chaque «*cycle d'horloge*», l'*unité de contrôle* lit une *instruction* du programme à partir de la *mémoire d'instruction* et envoie le résultat à l'*unité de traitement.* Celle-ci *décode* l'instruction et l'exécute sur une donnée, qu'elle peut conserver localement dans un registre ou ré-envoyer en mémoire. Ce *processus* reprend ensuite avec l'instruction suivante, dont l'adresse a été préalablement stockée dans un registre spécial local, jusqu'à la fin du programme. On constate donc un **modèle de calcul** intrinsèquement **séquentiel**.

Pour effectuer un calcul, un tel calculateur est obligé de faire des transferts incessants entre sa mémoire et son processeur. On voit donc apparaître une première limitation d'un tel fonctionnement, en l'occurrence, le *goulet d'étranglement* entre le processeur et sa mémoire. Ce débit ne peut augmenter *arbitrairement*, à cause de contraintes technologiques. Bien que, avec les progrès de cette technologie, on ait vu se concrétiser la notion de *hiérarchie mémoire*, qui limite les accès du processeur à sa mémoire centrale, en exploitant la notion de *localité des données* d'un programme. Le processeur est muni d'une "mémoire locale", appe-

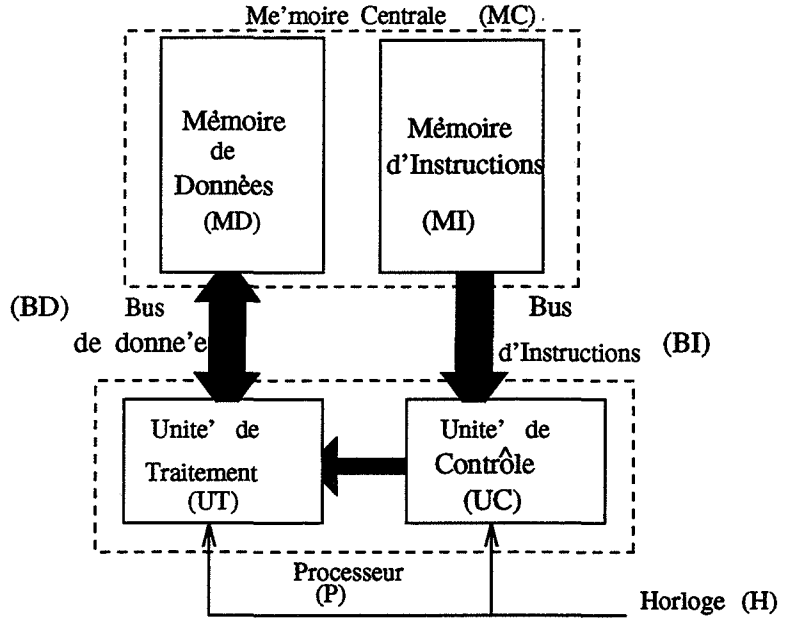


FIG. 1.1 – *Synoptique d'un calculateur de* VON NEUMANN

lée *mémoire cache*, de taille relativement petite, mais dont l'accès est beaucoup plus rapide que la mémoire centrale. Cette mémoire contient constamment, les données et les instructions les plus récemment accédées [30].

Les années 60 ont vu naître et se développer, d'une manière surprenante, la technologie des *circuits intégrés*. De manière simpliste, un circuit intégré peut être vu comme un graphe de connexions entre composants élémentaires (tels que des transistors, des diodes, des résistances, etc...) gravés dans une *plaquette de silicium* et dont la surface est de l'ordre du centimètre carré. Ce dispositif réalise une fonction donnée. Certains types de circuits (circuits logiques) permettent la réalisation de *fonctions booléennes*. Ce dispositif matériel est communément designé sous le terme de *puce*. Ces puces sont plus ou moins complexes, suivant le nombre de composants élémentaires qu'elles intègrent dans le silicium; on parle alors de *densité d'intégration* et le synoptique de VLSI[1], dérive de ce concept et correspond actuellement à une densité de l'ordre de $10^7$ éléments fonctionnels dans une puce [30].

Cette nouvelle technologie a permis la réalisation de concepts architecturaux qui étaient déjà connus et l'on a vu câbler au niveau du silicium, le concept de

1. Very Large Scale Integration : technologie de construction des circuits intégrés.

*microprogrammation*, de hiérarchie mémoire, de *registres d'index*, etc... . Cela a alors permis la réalisation d'ordinateurs[2] plus performants, plus compacts, plus fiables et moins chers [83]; pour aboutir finalement au début des années 70, à la réalisation du premier *microprocesseur* complètement contenu dans une puce de silicium, le 4004 d'INTEL [30].

Toujours dans un souci de gain de performances, les années 80 ont vu apparaître l'approche RISC[3] pour la construction des processeurs. En effet, dans l'approche initiale, i.e. les processeurs CISC[4], la technologie de construction des mémoires de l'époque, induisait des temps d'accès très lents; cette ressource était alors précieuse. Pour réduire le temps d'exécution total d'un programme, il fallait donc réduire au maximum les accès à la mémoire centrale. Pour cela, le processeur devait offrir un *jeu d'instructions* et un *mode d'adressage mémoire* très variés et complexes, qui correspondaient en fait à chaque type d'instruction qui peut apparaître dans un programme. Cela évitait la décomposition d'une instruction en plusieurs sous-instructions qui impliquaient chacune un accès mémoire. Cependant, des analyses de codes de programmes avaient montré que, d'une manière générale, ce code portait plutôt sur un jeu d'instruction plus réduit [30]. Par ailleurs, la technologie de construction des mémoires ayant énormément progressée, pour aboutir finalement à une ressource banalisée, la philosophie a été inversée. Le processeur admet alors un jeu d'instructions et des modes d'adressage mémoire plus réduits, mais plus efficaces et plus rapides; c'est l'approche RISC. Notons toutefois que de nos jours la séparation entre les deux types de processeurs est rendue subjective, par les progrès justement de la technologie.

Sont apparus également, les processeurs de type VLIW[5] qui *répliquent* les unités de traitement pour exécuter plusieurs instructions en un même cycle d'horloge. En général, le nombre d'unités de traitement est de 2 à 4. Une instruction VLIW est une instruction longue composée par la concaténation d'instructions destinées aux $n$ unités de traitement. C'est le compilateur qui a la responsabilité de générer le code en format VLIW à partir du programme source. Cela, en analysant les dépendances entre données du programme [83].

Avec les processeurs VLIW, on voit donc apparaître les premiers pas, *implicites*, vers le parallélisme. En effet, ce mode de fonctionnement présente des analogies avec le fonctionnement des calculateurs parallèles SIMD. A la différence près que les instructions délivrées aux unités de traitement sont différentes les unes des autres [83].

Cette évolution de l'ordinateur amena l'élargissement, de plus en plus grand,

2. Toujours séquentiels.
3. Pour Reduced Instruction Set Computer.
4. Pour Complex Instruction Set Computer.
5. Pour Very Long Instruction Word.

de son domaine d'application vers des applications, de plus en plus volumineuses et complexes. Alors que sa structure matérielle sous-jacente, pour l'exécution de ces applications, est restée fondamentalement inchangée. Elle est restée attachée au modèle de fonctionnement de type VON NEUMMAN. Les progrès de la technologie ayant caché et retardé l'exploration d'autres voies de puissance de calcul.

L'autre approche dans cette recherche de puissance de calcul, basée sur des architectures nouvelles, part du principe que de toute manière on veut construire des ordinateurs dont les performances évoluent plus vite que la rapidité des composants élémentaires; c'est l'approche du parallélisme.

## 1.2 Les architectures parallèles

Dans les systèmes parallèles, le but primordial escompté est la recherche de la **puissance de calcul**. Ce paradigme exprime la possibilité de résoudre des problèmes de taille de plus en plus grande, en des durées de temps de plus en plus petites[6].

La figure 1.2 illustre une abstraction de la machine de VON NEUMANN de la figure 1.1, qui ne tient compte que des éléments fonctionnels et de l'ensemble des *flots.*

Sur cette figure, un premier *flot d'instructions (FI)*, provenant de la mémoire d'instruction, passe à travers l'unité de contrôle pour générer un *flot d'instructions contrôlées (FIC).* Ce flot, ainsi qu'un deuxième *flot de données (FD)*, provenant d'une mémoire de données, sont alors consommés au niveau de l'unité de traitement pour produire un *flot de données résultats.* Ce dernier est finalement envoyé à la mémoire de données. Ce fonctionnement étant régulé par un *flot de "tics" (FT)* provenant d'une horloge.

Partant d'une telle abstraction, pour aboutir à un modèle de calcul parallèle (qui définit une architecture parallèle), une *réplication* des éléments *FI* et *FD* est nécessaire. Ainsi, suivant que ces deux types de flot sont *simples (S)* ou *multiples*

6. Un autre objectif que l'on peut attendre du parallélisme est lié à la nature même d'un *modèle de calcul parallèle.* En effet, une approche intrinsèquement parallèle pour la résolution d'un problème donné, peut déboucher sur une solution originale et efficace de ce problème; à l'instar de l'approche séquentielle classique.

Une illustration de ce propos, est représentée par le modèle **Data-flow** [4]. Dans ce modèle, le parallélisme est implicite et le calcul est intrinsèquement parallèle. Le problème de la suite de HAMMING (ce problème consiste à générer, *"d'un seul coup"*, l'ensemble des entiers naturels s'écrivant sous la forme $2^i 3^j 5^k$ pour $i, j, k$ entiers naturels, dans un ordre *croissant* et sans *répétition*) s'exprime de manière tout à fait naturelle et aisée dans un langage Data-flow; donc dans une algorithmique data-flow [4]. L'expression du même problème dans un langage séquentiel n'est pas évidente [4].

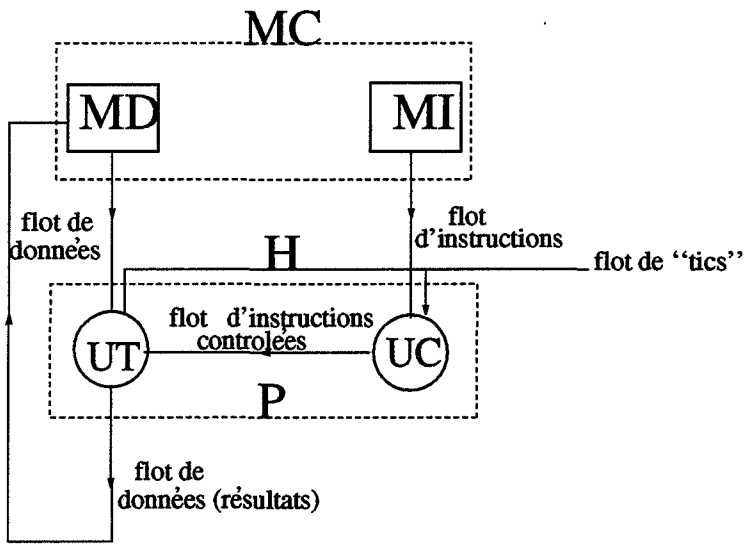


FIG. 1.2 – *Une abstraction du modèle de* VON NEUMANN *sous forme de flots*

*(M)*, on retrouve la classification dûe à M.J. FLYNN [32]. Cette classification distingue les modèles **SISD** (Pour, Single Instruction stream, Single Data stream), **SIMD** (Single Instruction stream, Multiple Data stream), **MISD** (Multiple Instruction stream, Single Data stream) et **MIMD** (Multiple Instruction stream, Multiple Data stream) :

1. SISD : il s'agit du processeur dans le sens commun, les instructions sont exécutées l'une après l'autre et elles agissent séquentiellement sur les données.

2. SIMD : par rapport au processeur précédent, on permet l'exécution d'instructions qui agissent identiquement sur plusieurs données à la fois, ce qui introduit le parallélisme. Le flot d'instructions provient d'une mémoire qui est commune à tous les processeurs.

3. MIMD : on introduit encore plus de parallélisme puisque la machine peut exécuter en même temps des instructions différentes sur des données différentes. Cela donne le plus de liberté au programmeur. Chaque flot d'instruction provient de mémoires de programme différentes ou pas.

4. MISD : l'exécution simultanée de plusieurs instructions sur un même flot de données introduit le parallélisme. Ce type de parallélisme correspond à la catégorie des machines pipelinées.

Nous nous intéressons aux architectures parallèles de type MIMD, à mémoire distribuée, dites à *parallélisme massif*. La machine comporte un nombre important de nœuds de calcul autonomes, possédant un processeur et une mémoire locale. Cet ensemble de nœuds sont interconnectés par un ensemble de liaisons point à point selon un *réseau d'interconnexion fixe*; i.e. le réseau d'interconnexion n'est pas reconfigurable. On suppose également que la machine satisfait à la propriété d'extensibilité[7].

## 1.3 Contrôle des architectures parallèles

L'exploitation de telles architectures pose des problèmes nouveaux par rapport à la machine séquentielle classique. En effet, dans son rôle de virtualisation de la machine physique en une machine virtuelle prête à être utilisée par l'utilisateur, le système d'exploitation d'une architecture parallèle doit offrir un ensemble de services spécifiques au parallélisme. Au traditionnel cycle de vie d'un programme séquentiel (i.e. compilation, édition de liens, chargement), s'ajoute de nouvelles fonctionnalités, dont dépendent étroitement les performances de l'application, dans le cas d'un programme parallèle. A notre point de vue, des plus importantes parmi ces tâches, on peut citer:

1. L'application doit être, tout d'abord, partagée en plusieurs tâches qui vont être exécutées en parallèle: c'est le problème du *partitionnement*,

2. L'ensemble des tâches ainsi obtenu, doit être ensuite affecté aux divers processeurs de la machine: c'est le problème de l'*allocation ou placement*,

3. Puisque les divers processus composants l'application ne seront certainement pas placés sur les processeurs de telle sorte que, deux processus qui ont besoin de communiquer sont toujours placés sur des processeurs voisins, il se pose le problème capital du *contrôle des communications et synchronisations.*

### 1.3.1 Le partitionnement

Le partitionnement d'un programme parallèle a pour but de préciser les parties du programme qui peuvent être exécutées en parallèle, et d'extraire un certain

7. Cette notion est une caractéristique du parallélisme massif. Elle exprime la possibilité d'étendre physiquement une architecture parallèle en nombre de processeurs dans le but de pourvoir à la puissance de calcul nécessaire à une application donnée, sans pour autant en modifier le code.

nombre d'informations sur la structure du programme. Chaque partie du programme correspond à une tâche qui va être exécutée séquentiellement sur un processeur. Plusieurs caractéristiques sont rattachées à une tâche dont :

- Son temps d'exécution séquentiel; i.e. sa taille,
- Les tâches avec lesquelles elle communique et le volume d'informations échangées,
- Les contraintes de précédence entre les différentes tâches, qui spécifient le degré de concurrence dans l'exécution du programme parallèle.

Le partitionnement d'un programme parallèle étant nécessaire, il se pose le problème important du "meilleur" partitionnement, qui minimise le temps global d'exécution de l'application. Outre les trois contraintes ci-dessus rattachées à chacune des tâches, d'autres critères interviennent dans la minimisation de ce temps, dont les coûts des communications. Le problème du partitionnement de programmes parallèles a été montré *NP-Complet* [85].

### 1.3.2 L'allocation et l'ordonnancement

L'allocation consiste à affecter les diverses tâches aux processeurs dans le but de minimiser le temps d'exécution parallèle du programme. Ce temps dépend du taux d'utilisation des processeurs et de la surcharge induite par les communications inter-processeurs.

De même, le problème de l'allocation n'est pas facile, et plusieurs contraintes interviennent dans la minimisation du temps d'exécution parallèle du programme. Par exemple, le problème de l'allocation d'un ensemble de tâches qui communiquent, ou qui admettent des contraintes de précédence a été montré *NP-Complet* [35], lorsque les tâches sont de tailles quelconques.

### 1.3.3 Le contrôle des communications et synchronisations

La communication et la synchronisation permettent aux processus d'une application de coopérer pendant l'exécution concurrentes de l'application. Ces deux concepts sont inhérents au parallélisme, dans le sens où dès qu'une application est partagée en un ensemble de tâches, dans le but d'une exécution concurrente, elles doivent communiquer puisqu'elles sont certainement *mutuellement dépendantes* [89].

Bien que notre intérêt ici est porté sur le **logiciel de contrôle** des communications, notons toutefois que la problématique de la communication se pose

également au niveau du réseau d'interconnexion processeurs-processeurs. En effet, ce dernier est à la base du mécanisme de coopération entre les processeurs, donc à la base de l'efficacité de l'exécution d'une application parallèle. Ce constat est rendu fort d'une part, dans le contexte d'un parallélisme massif qui implique un taux de communication élevé entre processus, et d'autre part, avec une puissance d'exécution élevée des processeurs qui implique que le réseau ne doit pas constituer un goulet d'étranglement dans l'échange d'information et pénaliser ainsi cette puissance. Plusieurs paramètres du réseau d'interconnexion influent sur les performances des communications, dont le degré des nœuds et le diamètre du réseau [25]. Une présentation de quelques architectures de nouvelles machines massivement parallèles avec leurs réseaux d'interconnexion et leurs systèmes de communication associés peut être trouvée dans [17].

Le logiciel de contrôle des communications, i.e. **le noyau de communication**, est l'une des composantes essentielles d'un *système d'exploitation* d'une architecture parallèle à mémoire distribuée. Ce noyau doit assurer principalement deux fonctionnalités.

Au niveau processus, le noyau de communication doit spécifier et gérer un ensemble de *protocoles* d'échange de données entre ces processus. Il doit également offrir un ensemble de routines de communication assez conviviales, qui constituent les points d'entrée des utilisateurs dans le système. Une bonne référence sur les divers modèles d'interaction entre processus d'un programme parallèle est l'article de Bal et al [7].

Au niveau processeur, le noyau de communication doit assurer une *virtualisation* du réseau d'interconnexion de la machine *cible* en un réseau complètement **connecté**. En l'occurrence, un réseau d'interconnexion où chaque paire de processeurs est reliée par un **chemin de communication**[8]

Cette virtualisation du réseau de processeurs est obtenue grâce à un **noyau de routage**, qui constitue la première couche logicielle du noyau de communication. Le noyau de routage assure la fonction de **routage des messages** entre toute paire de processeur *(source, destination)*. Au niveau d'un processeur, le **routage** est le terme utilisé pour décrire la procédure de décision, à travers laquelle le processeur choisit un (ou plusieurs) processeurs voisins, pour envoyer un message dans son chemin vers sa destination finale [89].

Dans le but de la virtualisation du réseau de processeur, divers *modes de communication* peuvent être à la base du noyau de routage. C'est par exemple une **reconfiguration dynamique** [76, 94], ou un **protocole de diffusion** [27] ou encore un **acheminement** depuis **un émetteur** jusqu'à **un récepteur** [48]. En ce sens, un mode de communication basé sur un protocole de diffusion consiste à

8. La réalisation d'un tel réseau par câblage, étant impossible à cause du coût de celui-ci, de plus l'on perdrait le concept d'extensibilité de la machine physique.

diffuser l'information à transmettre à partir du nœud source vers l'ensemble des nœuds du réseau. Cette information est alors soit acceptée, dans le cas d'un nœud destinataire, soit ignorée pour un nœud non destinataire. Alors qu'un mode de communication basé sur un acheminement entre un couple **(émetteur , récepteur)**, consiste à envoyer l'information à transmettre à partir du nœud source vers l'unique noeud destination suivant un ou plusieurs chemins de communication.

Ainsi, le mode de communication représente un point important dans la réalisation de noyaux de routage. Cependant, un bon choix consiste à baser la première couche logicielle sur un service de type (un émetteur, un récepteur); ceci constitue en fait une *brique de base nécessaire.* Sur ce service, d'autres modes de communication peuvent ensuite être construits [27]. En particulier, pour optimiser l'utilisation des ressources physiques de la machine, il est utile d'offrir un service qui concerne plusieurs processeurs en même temps; on parle alors de **communication globale** [26, 87]. Parmi ces schémas de communication, on peut citer :

- **La diffusion :** C'est le schéma de communication globale le plus utilisé. La diffusion consiste à envoyer un même message à tous les processeurs à partir d'un émetteur donné. On peut trouver une bonne étude de ce schéma dans [27, 70].

- **L'échange total :** C'est une diffusion simultanée à partir de tous les processeurs. Chaque processeur envoie son message vers l'ensemble des autres processeurs.

- **La distribution :** C'est une diffusion à partir d'un processeur donné d'un ensemble de messages destinés chacun à un processeur distinct. Chaque processeur qui reçoit son message ne le rediffuse pas, puisqu'il lui est proprement destiné.

- **La multidistribution :** C'est une distribution à partir de tous les processeurs simultanément. Chaque processeur envoie un message particulier à chaque autre processeur.

En effet, l'étude d'algorithmes classiques montre que ce genre de communication est très utilisé. C'est par exemple le cas en algèbre linèaire [58] et en traitement d'images [71].

## 1.4 Problématique et apport de la thèse

Lorsque le mode de communication est de type *(un émetteur, un récepteur)*, la fonctionnalité de virtualisation du réseau de processeurs en un réseau com-

plètement connecté que doit assurer le noyau de routage, est obtenu par le biais d'une **stratégie de routage**. La stratégie de routage spécifie les chemins de communication pour toute paire de processeurs (source, destination).

Une telle stratégie admet une incidence directe sur les performances du noyau de routage, et par là sur les performances du noyau de communication. Cette incidence est bien spécifiée par un ensemble de **critères de routage** que cette stratégie doit satisfaire. De manière générale, on distingue d'une part, des *critères de correction* qui doivent être *nécessairement* assurés et d'autre part, des *critères d'efficacité* à intégrer si possible. Les critères de correction sont :

- La spécification d'au moins un chemin de communication entre toute paire de processeurs,
- L'évitement de l'interblocage.

Les critères d'efficacité sont principalement représentés par :

- La définition de plusieurs, voire tous les chemins de communication optimaux entre les paires de processeurs (*source*, *destination*),
- L'utilisation d'un espace mémoire indépendant de la taille du réseau par rapport au nombre de nœuds de celui-ci,
- L'utilisation de l'ensemble des liens de communication du réseau.

L'objectif de l'étude présentée dans ce rapport est la conception de stratégies de routage, pour les réseaux de processeurs, qui satisfont à la fois les critères de correction et d'efficacité. Nous proposons une méthode de conception de stratégies de routage permettant, par une démarche incrémentale, de satisfaire les deux types de critères : **la communication multi-niveaux.**

Le caractère incrémental de cette méthode vient du fait qu'elle part d'un schéma de communication donné, qui doit satisfaire *"efficacement"* les critères de correction : **le schéma de communication primaire.** La méthode permet ensuite l'intégration des critères d'efficacité, pour l'obtention d'un **schéma de communication général** correcte et efficace.

Dans un premier temps, nous nous intéressons à la communication multi-niveaux basée sur le **routage par cycle eulérien** [72] comme schéma de communication primaire. Nous posons le *problème de l'efficacité* d'une telle approche de conception de stratégies de routage, dans les réseaux de processeurs.

Pour répondre au problème posé, nous appliquons le schéma de communication général ainsi obtenu à un réseau particulier : **la grille torique.** Au travers des critères de routage de *trois algorithmes*, nous répondons ensuite au problème posé.

Du fait que le routage par cycle eulérien utilise un *parcours eulérien* du réseau d'interconnexion comme *structure de routage*, du fait également que l'implantation de la communication multi-niveaux que nous avons réalisée est *distribuée*, nous proposons un **algorithme distribué** pour le calcul d'un tel cycle dans un réseau de processeurs.

Nous nous intéressons ensuite à la communication multi-niveaux basée sur un schéma de communication primaire de type *e-cube*. De même, nous posons le problème de l'efficacité d'une telle approche, et appliquons alors le schéma général ainsi obtenu à la grille torique. Là encore, l'algorithme de routage obtenu est correct et satisfait beaucoup de critères d'efficacité.

Le schéma de communication primaire doit satisfaire **efficacement** les critères de correction. Dans ce but, nous proposons un schéma qui part d'une *spécification formelle du problème du routage sans interblocage dans un réseau de processeurs.* Pour mesurer l'efficacité de ce schéma, nous le comparons au routage par cycle eulérien, sur la grille torique de dimension deux.

## 1.5 Organisation du mémoire

Le rapport est organisé en deux parties. Le chapitre 2 de la première partie, est un *état de l'art* sur les problèmes de routage dans les architectures parallèles à mémoire distribuée. Après avoir spécifié le problème du routage, nous présentons une synthèse des différentes stratégies de routage et les mécanismes de commutation de données qu'elles utilisent. Une synthèse des problèmes liés au routage ainsi que leurs solutions, présentées dans la littérature, est également donnée.

Le chapitre 3 de la première partie est consacré aux stratégies de routage. Dans ce chapitre, nous montrons que les critères de routage sont conflictuels et interdépendants.

La deuxième partie est organisée en cinq chapitres. Le chapitre 4 spécifie le modèle de routage que nous considérons dans ce rapport.

Le chapitre 5 est consacré à la communication multi-niveaux basée sur les deux schémas de communication primaire que nous avons évoqués ci-dessus. Dans ce chapitre, nous présentons les diverses applications à la grille torique.

Au chapitre 6, nous présentons l'implantation de la communication multi-niveaux basée sur le routage par cycle eulérien, que nous avons développée.

Le schéma de communication primaire que nous proposons est décrit au chapitre 7, avec l'application aux grilles toriques de dimension 2.

# Première partie

# Acheminement des Messages par Routage

# Chapitre 2

# Etat de l'art sur le routage des messages

## 2.1 Spécification du problème de routage

Un problème de routage de messages est la donnée d'un ensemble de $M$ messages munis chacun d'une adresse de destination $d_i$. Initialement les messages sont stockés individuellement parmi les $N$ nœuds du réseau et dans la majorité des cas on supposera que les destinations sont toutes différentes. L'objectif est alors d'acheminer ces messages vers leurs destinations respectives en minimisant au mieux les durées d'acheminement [65].

C'est là une formulation *canonique* du problème de routage, qui ne tient compte que de la fonction objectif, *la durée des communications* et de l'ensemble des messages en tant que données initiales. Elle ne rend pas compte des contraintes liées à l'implantation des solutions. En effet, pour un réseau de processeurs à mémoire distribuée, le routage implique le partage par les messages d'un ensemble de *ressources physiques* depuis les nœuds sources jusqu'aux nœuds destinations.

D'une manière générale, on peut classifier l'ensemble des ressources partagées en trois catégories :

1. Espace mémoire alloué pour le stockage des **messages utilisateurs**,
2. Espace mémoire alloué pour le stockage des **informations de routage**,
3. Ensemble des **liens de communication** du réseau.

Le partage de ces ressources induit alors des contraintes qui influent sur les temps de communication et nécessite de ce fait une gestion efficace. En particulier, le noyau de routage doit :

*Définir au minimum* un chemin de communication entre toute paire de nœuds. Ceci revient à assurer la propriété de **complétude** [57].

Une fois ce chemin défini, le noyau doit garantir l'arrivée des messages à leur destination ; c'est la propriété de **vivacité** [89, 74, 31].

Le noyau doit également assurer la progression des messages dans le réseau et éviter, à tout instant, une situation d'**interblocage** entre messages [23].

Outre ces propriétés qui doivent être nécessairement satisfaites, la longueur des chemins de communication doit être minimisée au mieux ; ce qui équivaut à permettre un "*bon*" **facteur d'élongation** des chemins de communication [78, 89].

L'espace mémoire alloué au stockage de l'information de routage et au stockage des messages utilisateurs, doit être *indépendant* de la taille du réseau par rapport au nombre de nœuds. L'algorithme permettant le calcul de l'information de routage doit avoir une bonne *complexité en temps et en nombre de messages* [89].

Le noyau doit également éviter les situations de **congestion**[1] au niveau des nœuds, qui bloqueraient momentanément la progression des messages dans le réseau. Il doit éviter la **famine**, c'est-à-dire qu'un message tourne indéfiniment dans le réseau sans jamais atteindre sa destination. Il doit permettre d'autre part que les messages soient injectés dans la *couche de routage* dans des délais raisonnables [74, 31], et de manière *équitable* entre les utilisateurs [89].

Il est également souhaitable que le **flot** des messages soit équilibré à travers l'ensemble des **canaux de communication** du réseau à tout instant [89, 31] et que la propriété d'**équité** entre les **délais de transmission** des messages soit vérifiée.

La **tolérance aux pannes** physiques doit être également assurée, pour obtenir un noyau **robuste**[2] [89].

L'ensemble de ces contraintes définit en effet, une spécification plus *concrète* du problème de routage dans les architectures parallèles à mémoire distribuée. Cette formulation permet d'affirmer que deux classes de *critères* gouvernent la conception d'un noyau de routage : les critères de **correction** que le noyau doit *nécessairement* assurer et les critères d'**efficacité** qu'il serait *souhaitable* d'intégrer. La satisfaction simultanée de tous ces critères est difficilement réalisable [89].

1. La littérature utilise également le terme de **contention**.
2. On dit qu'un système est robuste, s'il est tolérant aux pannes physiques.

## 2.2 Une classification des solutions

L'analyse des fonctionnalités attendues d'un système de routage de messages permet la distinction de quatre grands critères, pour la conception de tels systèmes.

En effet, la virtualisation du réseau de processeurs en un réseau complètement connecté, nécessite d'abord, la détermination d'une *stratégie de routage* et d'un ensemble de *mécanismes de routage* adéquats.

La stratégie est la réponse à la question : *"On veut router un message m d'un nœud $n_1$ vers un nœud $n_2$ ; quels sont les chemins possibles pour ce faire ? "*. Elle permet la spécification d'un ou plusieurs *chemins de communication* depuis un nœud source jusqu'à un nœud destination.

Les mécanismes répondent à la question : *"suivant les chemins calculés, comment acheminer le message ? "*. Ils définissent le mode d'acheminement des messages.

Viennent ensuite les problèmes de *dysfonctionnement* ainsi que la manière de les traiter ; c'est la *politique de traitement de ces dysfonctionnements.*

Le dernier aspect concerne le *type de routage* à implanter ; en l'occurrence, un routage *centralisé* ou *décentralisé*, et, un routage *statique* ou *dynamique.*

## 2.3 Les types de routage

Dans un routage centralisé, une *requête* de communication est transmise par le nœud source vers un nœud **central** en lui indiquant le nœud destination. Le nœud central établit alors un chemin entre les deux nœuds et initie la communication [76, 86].

L'inconvénient majeur de ce type de routage est, comme pour tous les systèmes centralisés, la panne du nœud central qui empêche alors toute communication. Adopté dans le cadre des premiers réseaux d'ordinateurs géographiquement répartis comme TYMNET [86], ce type de routage a été complètement abandonné au profit du routage décentralisé.

Dans un routage décentralisé ou **local**, chaque processeur prend la décision de routage seulement en fonction de conditions *locales* et établit lui-même le chemin vers le nœud destination selon un *protocole* donné [76, 86]. C'est le routage utilisé dans les architectures parallèles.

On peut également classifier le routage selon le caractère statique ou dynamique [19]. Un routage est dit **statique** ou **déterministe** lorsqu'il *fixe* une fois pour toutes les chemins de communication entre toute paire de nœuds $(source, destination)$. Le nombre de chemins pour un couple de nœuds (émetteur, récepteur) pouvant être quelconque. Formellement, ce type de routage peut

être modélisé comme un *plongement*[3] du graphe complet $K_n$[4] dans le graphe du réseau physique considéré[5] [87].

Ce routage offre alors la possibilité de router sur les plus courts chemins, mais présente le grand désavantage de ne pas toujours permettre une adaptation aux changements de trafic dans le réseau, ni d'utiliser efficacement les liens de communication. Il est par conséquent sujet à de fréquentes situations de **congestion** des nœuds du réseau. Par ailleurs, dans [16], BRAGG&*al* déconseillent ce type de routage lorsque :

1. le trafic est soumis à de fortes croissances soudaines,
2. les besoins de communication qui déterminent le trafic sont inconnus,
3. le réseau ne peut pas se configurer en fonction des besoins de communication.

Dans ces cas, bien que le délai de transfert des messages soit imprévisible, les auteurs préconisent le routage **adaptatif**, également appelé **dynamique** qui permet de pallier ces inconvénients [15]. Basé sur la propriété de complétude du noyau de routage, les messages sont acheminés *dynamiquement*, de nœud en nœud, en fonction de l'évolution du trafic des messages dans le réseau [55]. Ce type de routage suppose donc qu'il existe plusieurs chemins de communication entre tout couple de nœuds.

La caractéristique fondamentale d'un routage adaptatif et décentralisé est que l'envoi d'un message par un nœud, se fait selon une *politique de décision* basée entièrement sur des informations disponibles localement [74]. Par exemple, une politique de **dispersion** appliquée au niveau d'un nœud disposant de quatre messages qui sollicitent tous le même canal de sortie, transmettra un des messages sur ce canal, et *dispersera* les trois autres sur d'autres canaux [74]. De plus, une conséquence de ce type de routage est que deux messages émis par une même source vers une même destination en des instants $t_1$ et $t_2$ différents, n'empruntent pas forcément la même route.

Dans [36], GAUGHAN et YALAMANCHILI présentent une classification de ce type de routage, pour des réseaux d'interconnexion **réguliers** (hypercube, grille, tore, etc), selon trois types de protocoles :

3. Soient $G$ et $H$ deux graphes, un plongement du graphe $G$ dans le graphe $H$ est défini par la donnée d'une application $f$ des sommets de $G$ vers les sommets de $H$ et d'une application $P_f$ des arêtes de $G$ dans les chaines de $H$.

4. Le graphe complet $K_n$ est le graphe à $n$ sommets qui sont tous reliés entre eux.

5. En supposant que ce réseau admet $n$ nœuds.

• **Protocole avec retour arrière/progressif:** les protocoles *progressifs* permettent aux messages d'avancer dans le réseau vers leurs destinations et de n'exécuter des retours arrière que dans des cas limités. Par contre, les protocoles *avec retour arrière* effectuent une recherche *systématique* des chemins dans le réseau avec des retours arrière si nécessaire, en utilisant des informations de sauvegarde d'historique de chemins pour éviter la redondance.

• **Protocole avantageux/non-avantageux:** un protocole de routage est dit avantageux lorsqu'il n'utilise que des liens qui rapprochent le message de sa destination. A l'opposé, pour un protocole non avantageux, tous les liens sont des candidats potentiels.

• **Protocole complètement/partiellement adaptatif:** en raison de restrictions sur le routage, afin d'éviter l'interblocage, certains protocoles ne sont que *partiellement adaptatifs*. Dans le cas contraire, il sont *complètement* adaptatifs. Par exemple, un protocole avantageux et complètement adaptatif n'est jamais contraint dans le choix d'un lien avantageux par d'autres contraintes de routage.

Un autre type de routage conçu pour prendre en compte le changement du trafic dans le réseau, est le routage **aléatoire (randomisé)** introduit par VALLIANT dans [92, 93]. Ce mode de routage comporte deux phases,

1. **la phase de randomisation**: elle consiste à envoyer chaque message du nœud source vers un nœud intermédiaire choisi de manière aléatoire. On espère ainsi obtenir une *dispersion* relativement *uniforme* des messages sur l'ensemble des nœuds du réseau qui permettra d'éviter les congestions.

2. **la phase déterministe** : la seconde phase consiste à router, de manière déterministe, chaque message vers sa destination réelle.

Tous les algorithmes de routage basés sur le principe de la randomisation, suivent ce schéma en deux étapes [41, 66]. C'est le mode de routage proposé dans les réseaux de processeurs de type T9000 d'INMOS [56].

## 2.4 Les mécanismes de routage

Les mécanismes de routage déterminent les techniques d'acheminement d'un message depuis un nœud source vers un nœud destination. Ils concernent principalement le mode de **commutation des données** et l'ensemble des procédures de gestion du **flux de données** au niveau d'un nœud, en l'occurrence le **contrôle de flot**.

### 2.4.1 Les techniques de commutation de données

L'acheminement des données ne trouve pas ses origines dans les architectures parallèles. Ce problème apparaissait déjà dans les réseaux d'ordinateurs géographiquement répartis. A l'origine, ces réseaux d'ordinateurs utilisaient un mécanisme offert par la téléphonie, communément appelé **commutation de circuit physique**. Il consiste à établir un chemin physique entre le nœud source et le nœud destination, en effectuant un certain nombre de *commutations* au niveau des commutateurs du réseau et à ne libérer ce chemin qu'une fois le transfert terminé [88]. Aucun multiplexage des voies de communication entre différents messages n'est possible.

Depuis, plusieurs autres techniques ont été proposées que l'on peut classifier globalement en **commutation de circuit** et **commutation de message**, suivant que la commutation porte sur le chemin emprunté par un message ou sur le message lui-même.

- **La commutation de circuit.** C'est le principe de la téléphonie. On établit d'abord le chemin de communication, en réservant une suite de canaux depuis le nœud source jusqu'au nœud destination, le transfert du message commence ensuite. Ce chemin est obtenu à l'aide d'une requête envoyée par la source qui permet sa construction de nœud en nœud jusqu'à la destination. Celle-ci acquitte alors cette requête en renvoyant un *accusé de réception* pour permettre le début de transfert du message.

  Selon qu'au niveau des nœuds intermédiaires, l'interconnexion entre liens d'entrée et liens de sortie se fait physiquement ou logiquement, on parle de commutation de circuit *physique* ou *logique*. Mais dans tous les cas aucun multiplexage de canaux du circuit entre différents messages utilisateurs n'est possible. De plus, si cette technique est bien adaptée pour l'acheminement de très longs messages, le taux d'utilisation des liens reste faible à cause des temps de *silence* et de la procédure qui réserve le chemin, qui reste lente.

  C'est pour pallier ces inconvénients que la **commutation de messages** a été proposée.

- **La commutation de message.** Cette technique consiste à acheminer les messages de nœud en nœud jusqu'à leur destination. Au niveau de chaque nœud intermédiaire, le message est complètement stocké avant d'être envoyé sur le prochain canal de sortie.

  Cette technique peut nécessiter un espace mémoire important au niveau de chaque nœud pour stocker le message en entier. De plus, si le prochain lien au niveau d'un nœud intermédiaire est libre, le message est tout de

même mémorisé en entier ; d'où de longs délais de transmission globale des messages. Pour remédier à cette situation, la **commutation de paquets** a été proposée.

- **La commutation de paquets.** C'est également une commutation de type "stocker et envoyer", mais qui porte sur des fragments de même taille des messages à acheminer. Les messages sont donc découpés en *paquets* qui sont acheminés de nœud en nœud avec mémorisation et réexpédition.

  Cette méthode nécessite de ce fait moins d'espace mémoire au niveau des nœuds intermédiaires et permet un gain en temps de transmission dû au caractère "*pipeline*" de la transmission. D'un autre côté, les paquets étant acheminés individuellement, ils peuvent emprunter des chemins différents améliorant ainsi le temps global de transmission du message ainsi que le taux d'occupation des liens de communication. Cependant, ce découpage du message implique un réassemblage des paquets au niveau du nœud destination et par conséquent des procédures de contrôle.

- **Le "virtual cut-through".** Même si le découpage des messages en paquets permet une meilleure utilisation des liens, il demeure que ces paquets sont *systématiquement* stockés au niveau d'un nœud intermédiaire, même si le prochain lien de sortie pour l'envoi d'un paquet, est libre.

  La technique du "*virtual cut-through*" se propose de résoudre ce problème. Le message est débité en entier au niveau du nœud source, guidé par son en-tête, il construit son chemin de communication de lien libre en lien libre. Si au niveau d'un nœud intermédiaire tous les liens de sortie sont bloqués, le nœud amorce alors une procédure permettant de récupérer le message dans un tampon au fur et à mesure qu'il arrive.

  Ainsi, les premiers bits du message peuvent être au niveau du tampon pendant que les derniers sont encore au niveau du nœud source ; le chemin entre les deux étant complètement monopolisé pendant le temps du transfert. Nous avons donc un comportement de type commutation de circuit entre le nœud source et le nœud intermédiaire, et un comportement de type commutation de message entre le nœud intermédiaire et la portion de route qui reste à effectuer.

  La technique du "virtual cut-through" a été proposée par KERMANI et KLEINROCK dans [59]. Elle est analysée et comparée avec la commutation de message et la commutation de paquets dans le même article.

- **Le "wormhole".** Pour la technique du "*wormhole*" [23], le message est découpé en entités de tailles égales appelées *flits* pour *"flow control digit"*

qui correspondent à la taille de la queue d'un canal de communication. Le premier flit contient l'adresse destination et guide l'ensemble des autres flits du message.

Dès réception du flit adresse au niveau d'un nœud intermédiaire, celui-ci calcule le canal de sortie. Si le canal est libre, le nœud aiguille alors le flit en-tête vers ce canal de sortie qui permettra à son tour aux autres flits de le suivre dans cette direction. Si le canal est occupé, le flit adresse est alors bloqué au niveau de la queue du canal et bloque à son tour la progression des autres flits.

Le message progresse donc dans le réseau à la manière d'un "*ver de terre*". Il bloque l'ensemble des canaux déjà acquis lorsque la tête ne peut plus progresser au niveau d'un nœud intermédiaire. Il libère l'ensemble des canaux que vient de quitter le dernier flit du message, si l'ensemble des canaux en aval de celui-ci peut contenir le message en entier ou si une partie de celui-ci est déjà arrivée à destination. Par conséquent, une fois un canal affecté au flit en-tête, celui-ci n'est libéré que lorsque le dernier flit du message y passe.

Cette technique est propre aux réseaux de processeurs offrant une transmission *fiable*. Elle nécessite peu d'espace mémoire, seul le flit a besoin d'être stocké. Elle est de ce fait bien adaptée pour une intégration au niveau du silicium [23]. On peut trouver une synthèse sur les différents aspects du routage "wormhole" dans les réseaux réguliers dans [76].

- **Le "mad-postman" :** Notons également la technique dite "mad-postman", bien que spécifique à des réseaux réguliers où une notion de dimension peut être définie. Elle se comporte exactement comme la technique du "*wormhole*" en jouant en plus sur un facteur d'anticipation. Cela permet, d'une manière générale, un gain en temps de transfert du message avec une probabilité assez élevée.

  Le gain probable de cette technique se situe au niveau du flit "adresse". En effet, dès réception de celui-ci au niveau d'un nœud intermédiaire, il est directement émis sur le canal de sortie dont la dimension est la même que celle du canal d'entrée, les autres flits le suivant dans cette direction. Une fois le flit entièrement reçu, une procédure de contrôle est lancée permettant de vérifier si la dimension de sortie était la bonne. Dans le cas positif, l'ensemble des flits successeurs continuent d'être envoyés selon cette direction, sinon le nœud effectue une redirection du flit en-tête selon la bonne dimension [57].

On espère ainsi obtenir un gain dans le temps du transfert, puisque le message ne change de dimension qu'en un certain nombre de nœuds restreint devant le nombre total de nœuds qu'il doit traverser. Notons toutefois que cette technique génère un ensemble de flits errants pour lesquels une procédure d'élimination devra être mise en œuvre.

Une étude analytique et comparative des trois techniques, commutation de circuit, commutation de message (paquet) et "virtual cut-through", faite par KERMANI et KLEINROCK dans [60], prouve que, lorsque le nombre de canaux par voie de communication est bien choisi, la commutation de circuit est meilleure que la commutation de message ou le "virtual cut-through". Cela à condition que les messages soient de taille importante et que le chemin de communication soit long. Dans tous les autres cas, le "virtual cut-through" est meilleur que la commutation de circuit ou la commutation de message. Pour des petits chemins de communication et des messages de taille "modérée", la commutation de message est mieux adaptée que la commutation de circuit.

La tendance actuelle du mode de commutation des données dans les architectures massivement parallèles, est le "wormhole". En effet, depuis l'apparition des processeurs "routeurs" qui intègrent de plus en plus les fonctions de routage dans le silicium, cette technique de commutation des données est la mieux adaptée [76, 68].

### 2.4.2 Le contrôle de flot

Nous regroupons sous ce terme l'ensemble des procédures de gestion du flux de données en entrée ou en sortie au niveau d'un processeur du réseau. On distingue d'une manière globale trois types de contrôle de flot :

1. contrôle de **flot local**,
2. contrôle de **flot entre nœuds adjacents**,
3. contrôle de **flot source/destination.**

Le contrôle de flot local concerne la gestion des files d'attente au niveau de chaque nœud (politique de service pour gérer les arrivées et les sorties des messages sur les canaux de communication). Ce type de contrôle concerne également la résolution de problèmes induits par les mécanismes de routage. C'est par exemple l'élimination des flits errants dans le cas d'une technique de commutation de type "mad-postman" [57], la datation des messages pour lutter contre la famine dans un routage de type dynamique ou encore le calcul du nœud intermédiaire, suivant une loi de probabilité donnée, dans le cas d'un routage aléatoire.

On peut trouver une illustration d'un tel contrôle de flot dans [40]. Les auteurs donnent un ensemble de procédures de contrôle pour éliminer la congestion, lorsque le réseau de communication est soumis à un trafic intense. Le mécanisme de commutation utilisé étant la commutation de paquets.

Le contrôle de flot entre nœuds adjacents concerne par exemple la limitation du taux d'émission d'un processeur afin de ne pas dépasser la capacité de réception du nœud adjacent. Généralement ce type de contrôle est réalisé au niveau du câblage. Par exemple, NI et MCKINLEY observent dans [76] que pour les architectures massivement parallèles utilisant la technique du "wormhole", le délai de propagation d'une horloge de synchronisation fait qu'il est très difficile de distribuer une telle horloge sur un nombre important de processeurs. De ce fait, la transmission d'un flit entre deux nœuds adjacents utilise un protocole de *synchronisation automatique* (*"handshaking"*), réalisé au niveau du câblage.

Pour le dernier type de contrôle, qui s'effectue entre un nœud émetteur d'un message et un nœud récepteur, il s'agit par exemple du traitement d'une requête de chemin de communication et de son acquittement lorsqu'on utilise la technique de commutation de circuit.

Le contrôle de flot est clairement défini au stade de la réalisation même d'un noyau de routage. C'est à ce niveau que les besoins de gestion du flux de données apparaissent en fonction de l'ensemble des critères de conception du noyau de routage, initialement choisis.

Quelques illustrations des besoins en contrôle de flot peuvent être trouvé dans [41] pour le cas du noyau de communication de la machine MEGA, et dans [91, 90] pour le système PARX.

## 2.5 Les stratégies de routage

Une stratégie de routage peut être globalement définie par la donnée de deux composantes :

1. **une fonction de routage**; c'est l'élément de base. La fonction de routage calcule le ou les chemins reliant tout couple de nœuds, au sens du graphe sous-jacent au réseau d'interconnexion *physique*,

2. **un mécanisme de virtualisation**. Dans le but de satisfaire les critères de routage énoncés précédemment, un ensemble de structures logicielles et de traitements associés peuvent être adjoints à la fonction de routage. C'est la *virtualisation* des ressources physiques.

L'implantation de la stratégie de routage définit l'**algorithme de routage**.

Plusieurs modèles mathématiques de fonctions de routage ont été proposés dans la littérature; une présentation en est faite dans [72, 38]. Ces modèles se distinguent par les paramètres d'entrée de la fonction. L'un des modèles les plus utilisés est celui dont la fonction a comme paramètres d'entrée, le nœud destination $d$ et le **lien d'entrée** $l_e$ d'un message au niveau d'un nœud donné et qui délivre le **lien** ou les **liens de sortie** $l_s$:

$$f(d, l_e) = l_s,$$

L'avantage d'un tel modèle est qu'il permet de montrer les *relations de dépendance* qui existent entre liens d'entrée $l_e$ et liens de sortie $l_s$. Cette notion sera développée dans la section sur l'interblocage.

De manière conceptuelle, la représentation mémoire d'une fonction de routage est une *tabulation* au niveau de chaque nœud du réseau. Pour toute destination dans ce réseau et pour tout lien d'entrée, cette table contient une entrée indiquant le ou les liens de sortie.

Outre cet espace mémoire, l'algorithme de routage nécessite également de l'espace mémoire pour le stockage des messages. Cet espace correspond à l'ensemble des tampons alloués au niveau de chaque nœud pour le stockage temporaire des messages utilisateurs. Ce sont des *files d'attente* associées aux liens de communication.

Au nombre des contraintes qui gouvernent la définition d'une fonction de routage pour un réseau d'interconnexion, nous pouvons principalement citer:

1. la **complétude**: ce critère de correction d'un noyau de routage doit être en effet assuré au niveau de la fonction de routage. Le terme de **validité** est également utilisé dans la littérature,

2. l'**indépendance** vis-à-vis de la taille et de la topologie du réseau: la fonction de routage ne doit pas prendre en compte dans sa formulation des aspects liés à la taille ou à la topologie du réseau d'interconnexion. Ce critère est très important pour l'extensibilité de l'architecture physique de la machine,

3. un espace de stockage **constant**: l'espace mémoire utilisé pour la représentation de la fonction de routage et le stockage temporaire des messages utilisateurs doit être constant. En l'occurrence, indépendant de la taille du réseau en nombre de nœuds. Ce critère est également important pour l'extensibilité de l'architecture physique de la machine,

4. un routage sur **plusieurs** chemins **minimaux**: la fonction de routage doit définir plusieurs chemins minimaux entre tout couple de nœuds. Ce critère contribue à l'obtention des délais de communication optimaux,

5. l'**absence d'interblocage**: l'interblocage est un problème central dans les machines massivement parallèles. Il s'agit d'assurer qu'une situation d'*attente cyclique* de libération de ressources (liens de communication ou tampons de stockage de messages) ne se produise jamais.

Outre cette dernière contrainte, qui est une forme de tolérance aux dysfonctionnements logiciels, l'*algorithme de routage* doit être également tolérant aux pannes physiques.

Les critères qui viennent d'être décrits, sont généralement dépendants les uns des autres, parfois même conflictuels [89]. Par conséquent, mis à part quelques réseaux de communication ayant de bonnes propriétés topologiques, il est difficile qu'une fonction de routage les satisfasse tous à la fois, directement sur le réseau physique. Il est par exemple difficile qu'une fonction qui route sur les plus courts chemins et évite l'interblocage, utilise un espace mémoire constant [69]. De même, il est également difficile qu'une fonction qui assure à la fois un espace de stockage constant et l'absence d'interblocage route sur les plus courts chemins [72].

Pour satisfaire le maximum de critères, le mécanisme de *virtualisation des ressources* de la machine est généralement l'approche utilisée, pour aboutir à une *stratégie de routage* correcte et efficace. Ce mécanisme de virtualisation est une réplication virtuelle d'une ou plusieurs ressources physiques de la machine. C'est par exemple une réplication de chaque canal de communication physique en plusieurs canaux virtuels, ou de chaque nœud physique en plusieurs nœuds virtuels. Cette réplication virtuelle des ressources de la machine se traduit au niveau implantation par davantage d'espace mémoire et un temps de gestion des ressources répliquées.

C'est ainsi que, l'étude de la corrélation entre les différents critères de routage est en général abordée sous l'angle de leur incidence sur cet espace mémoire.

Par exemple, dans [78], UPFAL et PELEG étudient la corrélation entre l'espace nécessaire pour la représentation mémoire d'une stratégie de routage donnée et le *facteur d'élongation des chemins*, i.e. le rapport maximal, pris sur l'ensemble des paires de nœuds du réseau, entre, la longueur d'une route induite par la stratégie de routage et la longueur d'un chemin minimal du réseau entre la même paire de nœuds. Ils évaluent à $\Omega(n^{1+1/(2k+4)})$ la borne *inférieure* du nombre *total*[6] de bits nécessaires à la représentation de la stratégie de routage; pour un réseau d'ordre $n$ et un facteur d'élongation $k \geq 1$ de la stratégie.

FRAIGNIAUD et GAVOILLE montrent dans [33], qu'il existe une classe de réseaux d'interconnexion possédant une mémoire *locale*[7] d'*au moins* $\Omega(n^2)$ bits,

6. i,e, calculée sur l'ensemble des nœuds du réseau.

7. C'est l'espace mémoire nécessaire à la représentation de la stratégie de routage au niveau d'un nœud du réseau.

pour un facteur d'élongation d'*au plus* 2. En utilisant les graphes aléatoires de KOLMOGOROV, VITÁNYI et *al* [13] généralisent l'etude de la corrélation entre la longueur des chemins de communication et l'espace mémoire de représentation d'une stratégie de routage, pour cette catégorie de graphes[8].

Dans ce papier, les auteurs considèrent différents modèles vis-à-vis de la manière de la représentation mémoire des nœuds du réseau. Les différents cas sont :

1. Initialement[9], les nœuds du réseau n'ont aucune connaissance des étiquettes de leurs nœuds voisins et utilisent des *ports* étiquetés, associés aux liens adjacents, pour distinguer leurs voisins. Dans ce cas, deux sous-cas sont distingués :
   (a) l'étiquetage des ports est fixé et ne peut être changé,
   (b) l'étiquetage des ports peut être changé, avant le calcul de la stratégie de routage.
2. les nœuds du réseau connaissent les étiquettes de leurs voisins, et à travers quels liens, ils peuvent être atteints[10]

Vis-à-vis des étiquettes des nœuds, les auteurs distinguent les 3 cas suivants :

$\alpha$. les nœuds ne peuvent pas être re-étiquetés avant le calcul de la stratégie de routage,

$\beta$. les nœuds peuvent être re-étiquetés, mais par permutation sur l'ensemble $\{1, ..., n\}$,

$\gamma$. les nœuds peuvent être re-étiquetés de manière arbitraire.

Le produit cartésien de ces différents cas résulte en neuf modèles possibles. Pour chaque modèle, le tableau ci-dessous donne les bornes supérieures et inférieures de la taille mémoire de représentation d'une stratégie de routage, *de facteur d'élongation unitaire*, pour la classe des graphes de KOLMOGOROV :

8. Qui constituent une fraction d'au moins $(1 - 1/n^3)$ de la classe des graphes d'ordre $n$.
9. Avant le calcul de la stratégie de routage.
10. Autrement dit, l'étiquette associé à un port d'un lien de sortie correspond à l'étiquette du nœud voisin auquel ce lien est relié.

| | $\alpha$ | $\beta$ | $\gamma$ |
|---|---|---|---|
| Bornes supérieures | | | |
| cas 1(a) | $O(n^2 log\ n)$ | $O(n^2 log\ n)$ | $O(n^2 log\ n)$ |
| cas 1(b) | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ |
| cas 2 | $O(n^2)$ | $O(n^2)$ | $O(n\ log^2\ n)$ |
| Bornes inférieures | | | |
| cas 1(a) | $\Omega(n^2 log\ n)$ | $\Omega(n^2)$ | $\Omega(n^2)$ |
| cas 1(b) | $\Omega(n^2)$ | $\Omega(n^2)$ | $\Omega(n^2)$ |
| cas 2 | $\Omega(n^2)$ | ? | ? |

Notons également que la taille d'un *tampon* pour le stockage temporaire des messages utilisateurs dépend aussi essentiellement des mécanismes de routage.

### 2.5.1 Le routage avec des tables de routage compactes

Les réseaux *réguliers*, qui constituent les réseaux d'interconnexion des machines parallèles commercialisées, permettent une représentation plus simple que les tables, de la fonction de routage[11]. Dans ces réseaux, les fonctions de routage sont souvent de type *calculé* [36, 44, 76]. Au niveau d'un nœud, le routage d'un message arrivé par un lien d'entrée et à destination d'un nœud donné, consiste à évaluer une expression arithmétique donnant le lien de sortie [76]. Dans des cas précis où l'expression arithmétique est simple, celle-ci peut être directement intégrée dans le silicium. C'est le cas de l'algorithme "**e-cube**" décrit dans [23] et utilisé par le **Torus Routing Chip** [23] ainsi que par le **Direct Routing Module Chip** [42]. Cet algorithme a été étendu dans [67] afin de permettre un routage adaptatif et sans interblocage dans l'hypercube.

Cependant, les réseaux réguliers ne constituent pas l'unique et exclusive classe de réseaux de communication, appropriés pour l'exécution d'applications parallèles. En effet, ainsi que le notent bien NORMAN *et al* [77], pour de nombreuses applications, un réseau irrégulier est bien plus performant qu'un réseau régulier. La fonction de routage dans ce cas est tabulée. Par conséquent, l'espace mémoire nécessaire à sa représentation peut croître avec la taille du réseau, mettant ainsi à défaut la troisième contrainte citée précédemment.

Dans ce cas, plusieurs solutions ont été proposées dans la littérature pour réduire la taille des tables de routage.

11. L'idée du routage par table est un concept plutôt utilisé dans les grands *systèmes répartis* où le réseau de communication reliant les différents *sites* ne définit pas une topologie régulière au sens des réseaux de communication des architectures parallèles.

### 2.5.2 Le routage hiérarchique

L'une des premières méthodes proposées est le routage hiérarchique dû à KLEINROCK et KAMOUN [63], pour la réduction de l'information de routage dans les grands réseaux d'ordinateurs.

Cette technique consiste, de manière générale, à garder sur chaque nœud une information de routage complète pour un ensemble de nœuds "proches", et une information *synthétique* pour les autres nœuds du réseau [89]. Une organisation **hiérarchique** du réseau est utilisée pour cela :

- Un nœud est un *groupe* de *niveau* hiérarchique 0 : un **0-cluster**,
- Les 0-clusters sont regroupés en plusieurs 1-clusters,
- Plus généralement, les $(i-1)$-clusters sont regroupés en plusieurs $i$-clusters,
- Le dernier niveau de la hiérarchie, représente tout le réseau.

La table de routage d'un nœud contient une entrée pour chaque nœud dans le même 1-cluster, une entrée pour chaque 1-cluster dans le même 2-cluster, et plus généralement une entrée pour chaque $(k-1)$-cluster dans le même $k$-cluster.

L'avantage d'une telle organisation du réseau est qu'elle permet d'exploiter une notion de *"localité"* des communications [89]. En ce sens, à priori dans un réseau d'interconnexion, beaucoup de communications ont lieu entre des nœuds qui sont relativement à de petites distances entre eux [89].

Cependant, le problème de cette organisation est de trouver la meilleure *partition* du réseau en clusters de sorte à minimiser la taille des tables. Sachant qu'il faut respecter un certain nombre de contraintes comme la *connectivité* entre clusters de même niveau [12], la *distance maximale* entre deux nœuds du même cluster, le nombre de nœuds et le nombre de liens internes d'un cluster. Les auteurs prouvent que, pour un réseau à $n$ nœuds décomposé en $m$ niveaux, la table de routage de chaque nœud possède au moins $O(mn^{1/m})$ entrées.

Ainsi, bien que la taille de la table de routage soit réduite par rapport à une représentation classique, elle dépend tout de même du nombre de nœuds du réseau.

BOULOUTAS et GOPAL ont montré dans [11] que pour des graphes quelconques, le problème de la partition en clusters en tenant compte d'au plus deux des critères précédents est *NP-complet.*

12. Pour chaque niveau de la hiérarchie, le sous-graphe engendré par les clusters de ce niveau doit être *connexe.*

### 2.5.3 Le routage par préfixes

Le routage par **préfixes** (prefix-routing) a été introduit par BAKKER *et al* [6] pour minimiser l'information de routage dans des réseaux pouvant évoluer dynamiquement par ajout ou suppression de nœuds.

L'idée est d'attribuer à tout nœud $n$, une étiquette $\alpha(n)$ représentant un mot sur un alphabet $\Sigma$ contenant au moins deux symboles. Chaque lien est également étiqueté par un mot sur $\Sigma^*$ qui est un préfixe de quelques étiquettes de nœuds. Le routage se fait alors de la manière suivante: un message destiné au nœud $n$ est transmis sur le lien dont l'étiquette est le plus long préfixe de $\alpha(n)$.

Les auteurs montrent que tout réseau admet un routage par préfixes et proposent une méthodologie de construction d'une fonction de routage basée sur ce principe. Néanmoins, cette méthode de routage ne considère pas le critère d'absence d'interblocage dans sa formulation. Le routage obtenu n'assure donc pas automatiquement ce critère de correction. De plus, les étiquettes de nœuds peuvent devenir trop grandes, de l'ordre du diamètre du réseau, et dépendre ainsi de la taille du réseau.

### 2.5.4 Le routage par intervalles

La table de routage classique consiste à représenter l'information de routage dans chaque processeur $p$ par une relation $R_p \subset P \times L_p$, où $P$ est l'ensemble des processeurs et $L_p$ l'ensemble des liens de communication du processeur $p$. Un couple $(d, l) \in R_p$ indique qu'au niveau du processeur $p$, le premier lien sur un chemin menant à $d$ est $l$. Une autre façon de voir l'information de routage est de considérer la relation $R_p^{-1}$, inverse de $R_p$. $R_p^{-1}$ associe à chaque lien $l \in L_p$ l'ensemble $P_l$ des processeurs dont un chemin d'accès admet $l$ comme premier lien. On peut donc représenter l'information de routage dans le processeur $p$ par une table ayant autant d'entrées qu'il y a de liens dans $L_p$, l'entrée correspondant au lien $l$ contenant l'ensemble $P_l$ [89].

La procédure de routage consiste, pour une destination $d$, à rechercher le lien $l$ pour lequel $d \in P_l$ et à envoyer le message sur $l$. Dans le cas général, cette deuxième représentation de la fonction de routage occupera sensiblement le même espace mémoire que la représentation classique sous forme d'une table à deux entrées. Par contre il y a un surcoût pour la procédure de routage car maintenant au lieu d'un simple accès à une telle table, il faut procéder à une recherche séquentielle dans l'ensemble $P_l$ [72].

Un cas particulier, où l'on peut réduire la taille de l'information de routage et le temps de recherche est celui où les processeurs peuvent être *renommés*, en utilisant des noms appartenant à un ensemble *totalement ordonné* et chaque ensemble

$P_l$ est constitué d'éléments *consécutifs*. $P_l$ est alors un **intervalle**, son plus petit et son plus grand éléments représentent ses bornes. Dans ce cas pour représenter $P_l$ il suffit de stocker ses bornes et la recherche d'un élément dans l'intervalle se réduit à une comparaison avec les deux bornes. La taille de l'information de routage stockée dans un processeur $p$ est alors en $O(|L_p|)$.

Toute la problématique du routage par intervalles est alors : étant donné une fonction de routage pour un réseau donné, trouver une méthode de renommage des processeurs du réseau de telle sorte que, pour chaque processeur $p$ et chaque lien $l \in L_p$, l'ensemble $P_l$ soit totalement ordonné, i.e. $P_l$ soit un intervalle. Ce problème n'admet pas toujours une solution. Lorsqu'il en admet une, celle-ci satisfait peu des critères de routage parmi ceux cités précédemment, pour des réseaux quelconques [89, 38].

Le routage par intervalles a été défini, implicitement, pour la première fois par SANTORO et KHATIB dans [84] ; les deux auteurs n'ayant pas utilisé le terme de routage par intervalles. VAN LEEUWAN et TAN utilisent alors explicitement ce terme dans [64] pour montrer qu'en fait cette nouvelle méthode définit un nouveau *schéma de routage* permettant la réduction de la taille des tables de routage.

• **Méthode de** SANTORO-KHATIB. Soit $G = (V, E)$ le graphe connexe non-orienté qui modélise le réseau. L'ensemble des étiquettes est $[0, (n-1)]$ où $n$ est le nombre de nœuds du réseau. La méthode comporte deux étapes, l'**étiquetage**[13] des nœuds et l'étiquetage des liens. A partir d'un nœud $c$ *centre* du graphe[14], on construit un arbre couvrant en effectuant un parcours en *largeur* d'abord. Un parcours, en *profondeur* d'abord de cet arbre permettra ensuite l'étiquetage des nœuds et des liens.

Cette méthode fournit un *schéma d'étiquetage* **valide** et sans interblocage ; elle n'utilise cependant que $n-1$ liens dans un réseau à $n$ nœuds, et la racine de l'arbre constitue un goulet d'étranglement pour le routage des messages [89].

• **Méthode de** VAN LEEUWEN-TAN. On désigne un nœud *racine* $c$ à partir duquel on construit un arbre couvrant en suivant un parcours de type profondeur d'abord. Les nœuds et les liens sont étiquetés pendant ce parcours de façon à ce que les étiquettes soient attribuées dans un ordre croissant et en utilisant tous les liens.

Cette méthode assure un schéma d'étiquetage valide mais ne garantit pas

13. La littérature utilise le terme d'étiquetage plutôt que le terme de renomage.

14. Le centre d'un graphe est un sommet d'*écartement* minimum. L'écartement d'un sommet est la longueur du plus grand des plus courts chemins de ce sommet à tous les sommets du graphe [8].

l'absence d'interblocage [72].

Une autre méthode d'étiquetage sans interblocage valable pour des réseaux généraux est proposée dans [72]. Basée sur le routage par cycle eulérien, cette méthode induit plus d'une étiquette par nœud, elle en utilise le demi du degré du nœud. Les intervalles sont des tables de taille $O(|L_n|^2)$.

Une bonne synthèse des techniques de compactage de l'information de routage dans les réseaux de processeurs à mémoire distribuée est présentée dans [89, 72].

Dans [38], une étude théorique est présentée concernant la complexité mémoire pour la représentation d'une stratégie de routage. En particulier, il est étudié le problème de la *compacité* maximale[15] du routage par intervalles, lorsqu'on ne considère que des routages de plus courts chemins. Dans cette thèse, il est établi qu'il existe une famille de réseaux d'ordre $n$ et de degré maximum $d$ fixé, pour lesquels tout algorithme de routage ne peut être codé localement en moins de $\Theta(n \log d)$ bits; sur l'ensemble des nœuds du réseau. Ce résultat est alors interprété par le fait que les *tables de routage classique sont asymptotiquement incompressibles.*

## 2.6 Les principaux problèmes de routage

La conception d'un noyau de routage soulève un ensemble de questions relatives à la manière d'aborder les **problèmes** pouvant survenir lors de l'exécution d'une application donnée. Nous entendons ici, les cas de dysfonctionnement du noyau de routage, qu'ils soient *temporaires* ou *permanents*, au niveau de la *couche physique* ou de la *couche logicielle.*

Bien que une politique de **l'Autruche** soit assez souvent utilisée, à savoir qu'aucune mesure n'est prise ni *a priori*, ni *a posteriori* pour résoudre ces dysfonctionnements, deux techniques sont globalement envisageables pour traiter de telles situations:

• **Technique de prévention.** Cette technique consiste à prendre en compte explicitement le problème lors de la phase de conception du noyau. On *prévient* le problème, le rendant ainsi impossible.

• **Technique de détection-guérison.** Pour cette technique, aucune mesure n'est *a priori* prise pour éviter l'occurrence du dysfonctionnement. Le mode opératoire dans ce cas consiste à détecter celui-ci puis à le résoudre.

Les dysfonctionnements permanents sont relatifs à la propriété de *sûreté* du noyau de routage, qu'ils inhibent. Ils se manifestent sous la forme d'une **panne physique** ou d'un **état d'interblocage.**

15. i.e. de combien est-ce qu'on peut réduire au maximum la taille mémoire de représentation d'une stratégie de routage.

### 2.6.1 L'interblocage

L'interblocage est un problème *central* dans les architectures massivement parallèles à mémoire distribuée. Dans ce type d'architecture, le taux de communication entre processus utilisateurs est tellement élevé que la probabilité d'occurrence d'une situation d'interblocage est également élevée[16]. Ce problème est largement traité dans la littérature et beaucoup de solutions pour le résoudre ont été proposées.

D'une manière générale, l'interblocage est une situation où un message est en attente d'un événement qui ne se produit pas. Il peut donc apparaître à différents niveaux d'un noyau de communication et sous plusieurs formes. Dans [46], GUNTHER distingue au moins 6 types d'interblocage :

1. direct entre deux nœuds adjacents,
2. indirect qui implique plus de deux nœuds,
3. dû au réassemblage de paquets d'un même message,
4. dû aux messages d'acquittement,
5. dû à un schéma de priorité sur les messages,
6. dû à un utilisateur non prêt à recevoir son message.

Les interblocages *direct* et *indirect* sont les seuls qui se situent au niveau de la couche de routage. Les autres types interviennent plutôt dans les couches supérieures du noyau de communication.

L'interblocage direct est une dépendance mutuelle entre deux *processus* seulement [46]. L'interblocage indirect, qui nous interresse ici, en implique davantage : *c'est une attente cyclique entre n processeurs, chaque processeur $p_i$ attend de son voisin $p_{(i+1) \bmod n}$ qu'il libère une ressource, un tampon de stockage ou un canal de communication, pour pouvoir lui envoyer une donnée, un message, un paquet ou un flit.*

Dans les premiers travaux sur ce problème, la ressource critique était le tampon de stockage d'un message. En effet, le mode de commutation utilisé étant la commutation de messages (ou paquets) ou le "virtual cut through", les messages transportés par les canaux de communication étaient de taille relativement grande et l'état d'interblocage était relatif à une attente cyclique entre les tampons de $n$ processeurs. De plus, pour ce type de technique de commutation, la correspondance entre canal de communication et tampon de stockage de message n'est pas bijective.

16. Lorsqu'elle n'est pas prévenue.

Avec la commutation de type "wormhole", cette correspondance est bijective et implicite. A chaque canal de communication[17] est associé un tampon correspondant à la taille d'un flit, et réciproquement. De ce fait, pour ce type de commutation, une attente cyclique entre $n$ tampons de $n$ processeurs, est équivalente à une attente cyclique entre les $n$ canaux de communication correspondant aux $n$ tampons.

#### 2.6.1.1 Interblocage et graphe de dépendance

La figure 2.1 illustre une situation d'interblocage entre un ensemble de 8 processeurs. Cette figure décrit une situation où huit tampons, appartenant chacun à un processeur, contiennent chacun un message et forment un circuit de demande de requête de transfert. Dans une telle situation, aucun message des huit tampons ne peut progresser et le système se retrouve bloqué.

Nous avons représenté sur la même figure, la manière dont une telle situation peut être atteignable. En effet, pour chaque message $m_i$ stocké dans le tampon $t_1$ d'un processeur $p_i$ (pour $i = 1...8$), il suffit de considérer qu'à un pas d'exécution antérieur d'une unité, le message $m_i$ était stocké dans le tampon $t_2$ du processeur $p_{i-1}$, pour avoir au pas d'exécution $i$ la situation d'interblocage.

Ainsi, dans un réseau de processeurs, un état d'interblocage peut être caractérisé par les deux conditions que sont :

1. la présence d'un circuit dans la *demande* d'utilisation des tampons,

2. tous les tampons appartenant à ce circuit sont pleins.

La caractérisation ci-dessus est une condition *"a posteriori"*; elle caractérise un état d'interblocage existant dans le réseau de processeurs. Elle représente une condition **nécessaire** et **suffisante** à l'occurrence d'un interblocage.

C'est ainsi que les premières solutions, proposées dans la littérature, construisent une *structure de routage acircuitique* sur le réseau de processeurs [89], de sorte que la première condition n'est jamais satisfaite. Cette structure de routage est généralement basée sur un *mécanisme de virtualisation.* En chaque nœud du réseau, elle consiste en une réplication des tampons alloués à la couche de routage en plusieurs exemplaires. Le nombre de tampons par nœud est généralement dépendant de la taille du réseau. Nous détaillons dans le paragraphe 2.6.1.3 ce type de solutions.

Dans [23, 24], DALLY *et* SEITZ donnent une autre caractérisation, *"a priori"* d'un état d'interblocage dans un réseau de processeurs. Cette caractérisation est

17. Un canal étant une voie de communication unidirectionnelle.

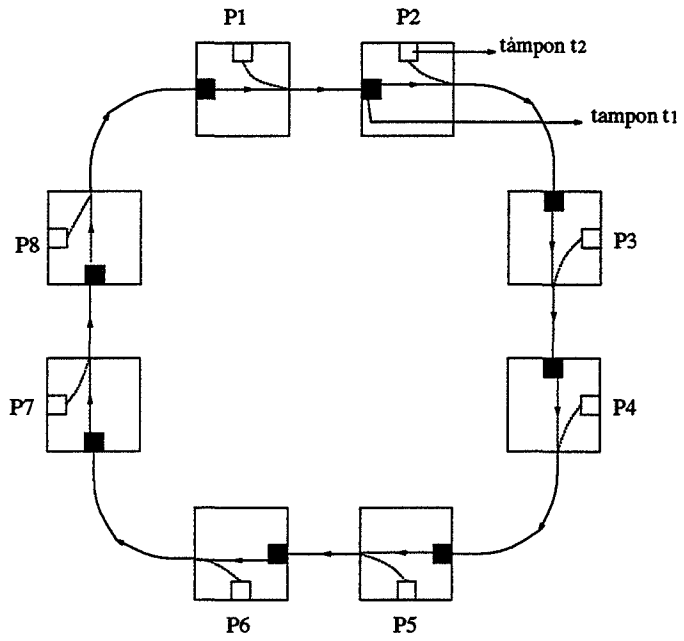


FIG. 2.1 – *Un état d'interblocage entre* 8 *processeurs*

basée sur la notion de **graphe de dépendance** induit par une fonction de routage sur un réseau :

**Théorème [23, 24]** : *Une fonction de routage $F$ sur un réseau d'interconnexion $I$ est sans interblocage* **ssi** *son graphe de dépendance $D_F$ est acircuitique.*

Le graphe de dépendance $D_F$ est un graphe orienté dont les sommets sont les canaux de communication de $I$ et les arcs sont les couples $(c_e, c_s)$ tels que le canal d'entrée $c_e$ *dépende* du canal de sortie $c_s$. La relation de dépendance entre un canal d'entrée $c_e$ et un lien de sortie $c_s$ est donnée par la définition suivante :

**Définition** : *Un canal d'entrée $c_e$ dépend d'un canal de sortie $c_s$ si $c_s$ est un successeur direct de $c_e$ dans un chemin de communication induit par $F$ sur $I$.*

Sur la base de cette caractérisation, les auteurs proposent une méthodologie de prévention d'interblocages en utilisant la notion de **canal virtuel**. La figure 2.2 illustre le principe de cette méthodologie, pour le cas d'un réseau en anneau, pour une direction de parcours. Le même principe est appliqué dans l'autre direction.

Cette méthode consiste à diviser un canal physique en un ensemble de **canaux virtuels**. Chaque canal virtuel admet son propre tampon et le temps de

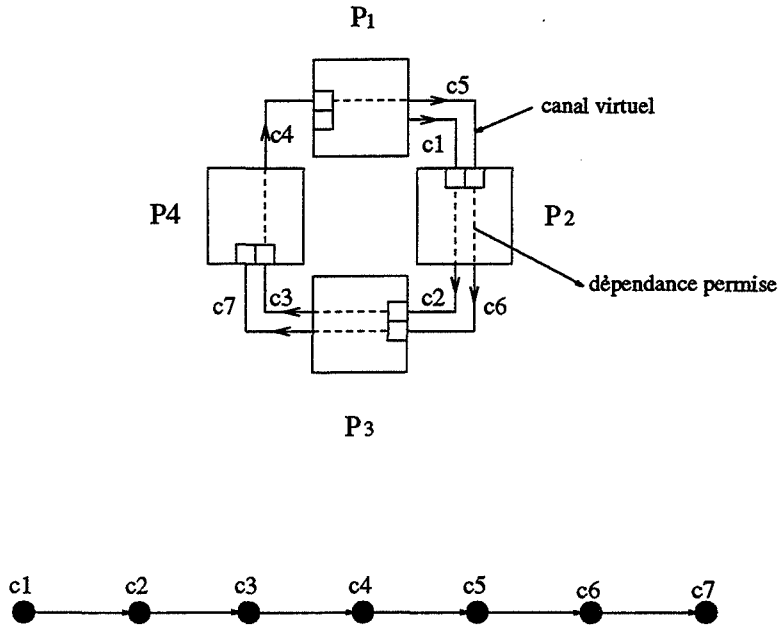


FIG. 2.2 – *Prévention de l'interblocage sur un anneau*

transfert sur le canal physique est divisé en tranches, une tranche par canal virtuel. L'absence d'interblocage est assurée par le fait de router sur une *spirale* de tampons qui passe par tous les nœuds. Le graphe de dépendance entre canaux virtuels de la stratégie de routage ainsi induite est acircuitique.

Dans [28], DUATO observe que la *condition d'équivalence* dans le théorème précédent (**ssi**) n'est valable que pour des fonctions de routage **statiques**. En fait, pour des fonctions de routage qui ne permettent qu'un *seul* chemin de communication entre toute paire de nœuds. L'auteur prouve que dans le cas des fonctions de routage **dynamiques**, qui permettent donc plusieurs chemins entre tout couple de nœuds, même si le graphe de dépendance $D_F$ admet un cycle, un état d'interblocage n'est pas *nécessairement atteignable* dans le réseau. En d'autres termes, la condition de l'existence d'un cycle dans le graphe de dépendance est *nécessaire* mais non *suffisante* à l'occurrence d'un état d'interblocage. L'argument est que lorsqu'un ensemble de chemins entre des paires de nœud (*source*, *destination*) induisent un cycle dans le graphe de dépendance, il suffit qu'il existe pour les paires de nœud en question d'autres chemins de communication qui n'induisent pas de cycle, donc qui n'induisent pas un interblocage.

En effet, la condition d'un graphe de dépendance acircuitique caractérise plutôt l'évitement d'une notion d'interblocage **potentiel** dans le réseau. En ce sens

qu'un cycle dans ce graphe de dépendance représente un état de blocage d'un ensemble de processeurs qui est *potentiellement* et non forcément atteignable.

Bienque étant une condition forte, l'absence de cycle dans le graphe de dépendance est nécessaire, lorsqu'on considère les critères d'efficacité qu'une stratégie de routage doit satisfaire.

En effet, l'évitement de la congestion implique qu'au niveau d'un nœud donné, pour un canal d'entrée d'un message, l'ensemble des canaux de sortie de ce nœud qui n'induisent pas un interblocage sont des canaux potentiels pour l'envoi du message. Si dans cet ensemble, il existe des canaux de sortie qui induisent un cycle dans le graphe de dépendance, on peut se retrouver dans la situation où, pour éviter la congestion on crée un interblocage. Le même argument est valable lorsqu'on considère la tolérance aux pannes physique. En effet, si l'on ne base pas la condition d'absence d'interblocage sur un graphe de dépendance sans cycle, on peut se retrouver dans la situation où pour détourner un chemin de communication à cause d'une panne de lien, on induit un interblocage.

Ainsi, d'une manière générale, pour avoir un noyau de routage *robuste* et *efficace*, il est préférable de baser la prévention de l'interblocage sur une condition forte telle qu'un graphe de dépendance acircuitique. Même si une telle condition peut se traduire par un "petit" espace mémoire de plus.

Mis à part les solutions dédiées à des architectures spécifiques, comme le routage *e-cube* dans le cas de l'hypercube binaire [23], ou le routage $XY$ dans le cas de la grille, il existe globalement deux méthodes de lutte contre l'interblocage :

- les méthodes de détection-guérison,
- les méthodes préventives.

#### 2.6.1.2 Les méthodes de détection-guérison

Elles procèdent en deux étapes : détection de l'état d'interblocage puis traitement de celui-ci. Comme le font remarquer GAMBOSI *et al* dans [34], afin de répondre à des critères d'efficacité, l'étape de détection doit être menée de façon distribuée. Tandis que l'étape de traitement doit garantir aux processeurs non impliqués dans l'interblocage de pouvoir poursuivre une activité normale.

C'est dans l'étape de détection que se distinguent les quelques rares méthodes dont fait état la littérature. Comme cette étape consiste en une recherche de cycle impliquant un interblocage, on distingue des stratégies de *détection explicites* et des stratégies *implicites.*

Les stratégies explicites se caractérisent par le fait qu'elles construisent systématiquement le graphe de dépendance des tampons dans lequel le cycle est à rechercher [34]. Par contre dans les stratégies implicites, une telle construction n'est pas nécessaire. La détection est faite à travers des stratégies d'allocation des

tampons d'émission de messages. De telles stratégies sont proposées par CHAN et YUM dans [14].

La phase de détection du cycle des dépendances est inhérente aux méthodes de détection-guérison. Explicite ou implicite, cette phase est coûteuse en temps et en ressources physiques et implique des messages de contrôle supplémentaires, qui surcharge le réseau de communication. Ces méthodes ne sont donc guère adaptées au parallélisme massif. Cette approche de résolution de l'interblocage n'a suscité que peu d'intérêt.

#### 2.6.1.3 Les méthodes préventives

Elles préviennent une situation d'interblocage en faisant en sorte que l'une au moins des deux conditions nécessaires et suffisantes à l'interblocage ne soit pas vérifiée. D'une manière générale, on définit un ordre, *spatial* ou *temporel*, sur l'ensemble des ressources de communication du réseau, de façon à ne pas permettre la formation de cycle. Plusieurs méthodes ont été proposées dans la littérature :

**(a) La méthode des graphes de tampons.** Historiquement, cette méthode est le premier type de solutions proposées dans la littérature. Elle regroupe en fait un ensemble de techniques qui sont toutes basées sur la construction d'un graphe *orienté*, *acircuitique* de *classes de tampons*. Parmi ces techniques :

1. la méthode de "comptage de vallées" [46, 86],
2. la méthode de "comptage de sauts" [43, 31],
3. la méthode de "comptage de sauts négatifs" [45],
4. la méthode du "class climbing" [3].

Les tampons d'un nœud sont répartis en plusieurs classes et un message n'est transmis d'un tampon vers un autre que s'il y a un arc de la classe du premier tampon vers la classe du second tampon. De même, on remarque que la prévention de l'interblocage est basée sur le principe de virtualisation. En effet, au niveau de chaque nœud, on réplique les tampons alloués à la couche de routage en plusieurs exemplaires.

Le grand problème de ces méthodes est le nombre de tampons nécessaires au niveau de chaque nœud. Ce nombre croît en fonction du nombre de nœuds du réseau. Aussi, ces méthodes représentent des techniques de gestion de ressources répliquées et nécessitent l'ajout d'une fonction de routage, qui généralement consiste en un routage sur les plus courts chemins.

**(b) La méthode de** ROSCOE **[80].** Cette méthode est basée sur des règles d'admission des messages dans la couche de routage.

Dans le cas d'un réseau en anneau unidirectionnel, chaque nœud est muni de deux tampons. L'absence d'interblocage est assurée par la contrainte qu'un nœud n'accepte un message venant d'un utilisateur, que si ses deux tampons sont vides.

Dans le cas d'un réseau quelconque, l'auteur propose de reconstruire la structure cyclique de routage soit par l'utilisation d'un cycle *hamiltonien* si le graphe en admet un, soit par l'utilisation d'un circuit hamiltonien virtuel obtenu à partir d'un arbre couvrant dans lequel on introduit des nœuds virtuels.

L'absence d'interblocage est alors assurée par les règles de routage suivantes :

- Un nœud n'accepte un message venant d'un utilisateur que si ses deux tampons sont vides,
- Les liens qui ne sont pas dans le cycle peuvent être utilisés à condition que les messages qu'ils transportent viennent juste d'être injectés dans la couche de routage.

Cette technique nécessite un contrôle lors de l'injection des messages au niveau de la couche de routage. Elle est essentiellement dynamique et ne peut donc être utilisée dans un routage déterministe. Il est également important de noter que la prévention de l'interblocage est basée sur le principe de virtualisation.

**(c) La méthode des graphes virtuels.** La méthode des graphes virtuels [57], consiste à diviser le graphe associé au réseau de communication en un ensemble de graphes virtuels, orientés, acircuitique, indépendants et disjoints. Chaque graphe virtuel dispose de sa propre classe de tampons. La prévention de l'interblocage est assurée par l'injection de tout message dans le graphe virtuel adéquat en fonction de son nœud source et de son nœud destination. Le message demeure dans ce graphe jusqu'à destination.

Le problème de cette méthode est qu'à l'exception de quelques réseaux réguliers (grille et tore $2D$), il n'y a pas de méthode générale pour la construction des graphes virtuels. Là encore, l'interblocage est résolu à l'aide de la technique de la virtualisation.

**(d) La méthode des canaux virtuels.** C'est la méthode évoquée dans le premier paragraphe de cette section [24, 23].

Elle consiste à multiplexer sur un même canal physique plusieurs canaux virtuels ayant chacun sa propre file d'attente. Cette *virtualisation* du canal physique en plusieurs canaux virtuels est obtenue en partageant le temps de transfert sur le canal physique en tranches, chacune réservée à un canal virtuel. L'absence d'interblocage est assurée du fait d'un graphe de dépendance entre canaux virtuels

acircuitique.

Le problème que pose cette méthode est le nombre de canaux virtuels nécessaires par canal physique. Les auteurs prouvent que deux canaux sont suffisants pour les *grilles toriques* $2D$ et trois pour le *Cube-Connected Cycles* $3D$.

Une méthode de construction d'un algorithme de routage sans interblocage par canaux virtuels, pour une *fonction de routage* et un réseau d'interconnexion donnés, est décrite dans [53].

**(e) La méthode de l'arbre couvrant** [91]. A partir d'un nœud racine $r$, on construit un arbre couvrant en suivant un parcours en largeur d'abord. A chaque nœud du réseau on associe son niveau, i.e. sa distance par rapport à la racine. Un canal est dit descendant s'il relie un nœud de niveau $i$ à un nœud de niveau $i+1$. Il est dit ascendant dans le cas contraire et horizontal s'il relie deux nœuds de même niveau.

La méthode de routage consiste à observer les **règles de dépendance** suivantes :

1. un canal descendant ne peut dépendre d'un canal ascendant, i.e. un canal ascendant ne peut être successeur direct d'un canal descendant, dans un chemin de communication,

2. un canal horizontal ne peut dépendre d'un canal ascendant,

3. il ne peut y avoir de cycle de canaux horizontaux dépendants.

On montre [91] que ces *règles* induisent une *fonction de routage* sans interblocage. Cependant, cette fonction n'assure pas un routage sur les plus courts chemins. Là par contre, la prévention de l'interblocage n'est pas basée sur le principe de virtualisation. En effet, on ne réplique pas les canaux physiques en canaux virtuels et un seul tampon est associé à chaque canal.

**(f) La méthode du cycle eulérien.** MUGWANEZA *et al* décrivent dans [73] une méthode de routage pour réseaux configurés statiquement, qui évite l'interblocage. Cette méthode est valable pour tout réseau dont les nœuds ont un nombre pair de liens, elle nécessite un seul tampon par canal et assure sous certaines conditions un routage sur les plus courts chemins.

En supposant que les liens de communication soient bidirectionnels, l'idée est d'exhiber un cycle *eulérien* associé au graphe du réseau d'interconnexion. Une fois le cycle calculé, on extrait de celui-ci un lien donné et on route selon des **règles de routage** sans interblocage sur le *chemin eulérien* ainsi obtenu. Le graphe de dépendance associé à ce chemin ne contient pas de cycle [72]. En pratique, c'est

plutôt un chemin eulérien qui est utilisé et le problème de l'arc à extraire ne se pose pas.

La condition sur la parité des liens assure l'existence d'un cycle eulérien. Dans le cas d'un réseau quelconque -qui n'assure donc pas forcément cette condition- les auteurs proposent de diviser un lien physique en un nombre pair de liens virtuels.

De même, cette méthode n'est pas basée sur le principe de la virtualisation pour l'évitement de l'interblocage. Elle admet de meilleures propriétés de routage que la méthode précédente [72]. En particulier, elle permet davantage de **dépendances** entre canaux de communication.

En effet, la notion de dépendance entre deux canaux de communication successifs [23] est à la base des méthodes préventives, qui ne nécessitent qu'un seul tampon par canal, et par là, qui n'utilisent pas la technique de la virtualisation. Aussi, plus une méthode de routage permet de dépendances, meilleures seront ses qualités.

Une approche générique consisterait donc, étant donné un réseau de communication, à considérer plutôt la structure de routage qui contient toutes les dépendances entre canaux de communication, pour en extraire un ensemble qui n'induit pas un interblocage. C'est l'approche que nous proposons dans le chapitre 7 de la deuxième partie. Nous considérons pour cela, le graphe adjoint [8] du graphe d'interconnexion des processeurs. En effet, ce graphe représente le support de toutes les dépendances entre canaux de communication. Au delà du problème de l'interblocage, cette approche permet d'intégrer d'autres critères de routage.

### 2.6.2 Les pannes physiques

Dans les architectures massivement parallèles, la *probabilité* d'occurrence d'une panne croît avec le nombre de processeurs dans la machine [29]. Il est donc important, lors de la conception d'un système de communication pour une telle machine, que la tolérance aux pannes physiques soit prise en compte.

On dit qu'un système est **tolérant** aux pannes physiques si, étant donné un ensemble de ressources physiques *défectueuses*, il existe dans celui-ci un moyen de *recouvrement* de ces pannes, de sorte à émuler un fonctionnement normal.

Dans la littérature, la tolérance aux pannes est généralement abordée sous l'angle de la défaillance d'un processeur plutôt que celle d'un lien de communication [12, 29]. Ceci à juste raison, puisque la probabilité de défection d'un processeur est plus grande que celle d'un lien de communication [29]. Par ailleurs, la panne d'un lien peut être modélisée comme une panne de processeur [29].

La mise en œuvre de la tolérance aux pannes nécessite une étape de *détection* et *diagnostic*, puis une étape de traitement de la panne. Toutes les méthodes de détection et diagnostic sont "ad hoc". Il n'existe en général pas de méthode

générique pour la phase de traitement, outre la *reconfiguration* du réseau d'interconnexion (lorsque cela est possible) de sorte à le rendre *isomorphe* au réseau initial [29].

Dans le cas spécifique de la couche de communication, les méthodes de traitement consistent à *contourner* tout chemin qui passe par un composant (lien de communication et/ou processeur) défectueux. Ces méthodes varient d'un réseau d'interconnexion à un autre. Pour des raisons de taux de réalisation, soit comme prototypes soit comme produit commercialisés, le traitement des pannes dans les réseaux réguliers (hypercubes, grilles, tores) est le plus étudié dans la littérature [69, 62, 44]. Il consiste à exploiter les propriétés topologiques régulières de ces réseaux.

Un algorithme illustratif de cette approche est proposé par KIM et SHIN dans [62], pour des réseaux de type hypercube. Il s'agit d'une extension de l'algorithme "e-cube" permettant un routage robuste, à condition qu'il existe dans un hypercube $H_n$ défectueux[18] au moins un hypercube $H_{n-2}$ non défectueux. Sur la base de cette hypothèse, les auteurs distinguent quatre cas possibles. Un nœud non défectueux doit alors correctement déterminer dans quel cas est le système, ainsi qu'à quel $H_{n-2}$ il appartient. La stratégie de routage consiste ensuite à observer les règles suivantes :

1. le routage à l'intérieur d'un sous-cube $H_{n-2}$ robuste utilise l'algorithme "e-cube",

2. lorsque le chemin entre un couple de nœuds passe par un nœud défectueux, une recherche dans une table de routage locale permet de détourner ce chemin.

### 2.6.3 La congestion, la famine et le refus des messages

Les dysfonctionnements temporaires sont de durée limitée. Ils sont relatifs aux propriétés de *vivacité* et d'*équité* et se manifestent principalement sous forme de **congestion**, de **famine** et de **refus des messages** dans la couche de routage.

La congestion est un état du réseau de communication, où un ou plusieurs nœuds ne permettent plus, momentanément, une *progression régulière* des messages dans le réseau. Ces nœuds sont appelés nœuds de congestion (*"hot spot"* en anglais). Au niveau de chaque nœud de congestion, les messages arrivant sur des liens d'entrée différents et sollicitant, sans succès, le même lien de sortie occupé, sont temporisés au niveau des tampons associés aux liens d'entrée et bloquent, de ce fait, l'arrivée de nouveaux messages sur ces liens d'entrée. La congestion

18. Un hypercube de dimension $n$ qui contient au moins un nœud défectueux.

apparaît généralement dans un routage statique où le choix des chemins de communication est fixé d'avance.

La famine est une situation où un ou plusieurs messages tournent indéfiniment[19] dans le réseau, sans atteindre leur destination [36]. Potentiellement, ce type de comportement peut être observé dans un routage adaptatif, à cause du caractère non déterministe de ce dernier.

Le refus des messages est une situation où la couche de routage n'accepte plus de nouveaux messages utilisateurs. Elle est causée par la saturation au niveau d'un nœud du réseau de l'espace mémoire alloué au stockage des messages en transit. En effet, s'il arrive qu'en raison d'un *afflux* permanent de messages intermédiaires au niveau de ce nœud, tous ses tampons internes soient pleins, il ne peut injecter ses propres messages.

Les trois dysfonctionnements qui viennent d'être décrits, sont dus à une mauvaise conception du noyau de routage. Ils peuvent être évités par des procédures locales de contrôle du flux de données. Dans le cas de la congestion, celles-ci se traduisent par le biais d'un routage dynamique [41, 55, 74]. Les messages dont la requête n'a pas été satisfaite ne sont pas temporisés mais envoyés sur d'autres liens de sortie, susceptibles ainsi de rallonger les chemins de communication.

Notons toutefois que, sans contrôle, ce type de routage peut inéluctablement conduire, pour certains messages, à une situation de famine. En effet, l'éloignement répété d'un message de sa destination peut aboutir à une errance sans fin de celui-ci. Ce qui n'est pas le cas dans un routage statique, où un message est acheminé de façon déterministe vers sa destination.

Le refus des messages dans la couche de routage est également une situation qui apparaît plutôt dans un routage dynamique [74]. Ainsi, ce type de routage nécessite un contrôle strict pour prévenir globalement chacun des trois types de dysfonctionnement. D'autant plus que c'est un type de routage de plus en plus utilisé dans les architectures massivement parallèles [36, 28, 69, 76]. Ce contrôle est généralement exercé par le biais d'un schéma de priorité sur les messages circulant dans la couche de routage. C'est une telle approche que décrivent NGAI et SEITZ dans [74] pour la mise en œuvre d'un modèle général de routage dynamique efficace.

La méthode est inductive. On suppose initialement un *environnement statique*; chaque nœud ne pouvant avoir injecté dans le réseau qu'un seul message au plus. De ce fait, à tout instant, le nombre de messages circulant dans la couche de routage est fini. Sur la base de cette hypothèse, on construit un *ordre linéaire de priorités* assignées aux messages. La priorité $p$ assignée à un message correspond à l'*identificateur* $n$ du nœud émetteur de celui-ci. Lors d'un *conflit d'accès*

19. Dans le pire des cas.

à un canal, le message de priorité la plus élevée acquiert le canal :

$$p_1 > p_2 \iff n_1 < n_2 \quad (1)$$

Les auteurs montrent que cet ordre de priorités évite la contention ainsi que la famine dans un environnement statique.

Ce schéma de priorité ne permet pas qu'il y ait deux messages émis par le même nœud, en circulation dans le réseau au même temps. Ce qui est restrictif et irréaliste. Pour pallier cette contrainte, on définit un second ordre de priorités. La priorité assignée à un message correspond à sa *distance courante d* par rapport au nœud destination. Elle est donc fonction de sa position relativement à la destination. Lors d'un conflit d'accès à un canal, le message de plus courte distance l'acquiert :

$$p_1 > p_2 \iff d_1 < d_2 \quad (2)$$

De même, les auteurs montrent que dans un environnement statique, ce schéma de priorités évite la contention ainsi que la famine des messages pris dans leur ensemble, i.e. ce schéma assure une *progression globale* des messages mais ne garantit pas une progression de chaque message *individuellement.*

Il est intéressant de noter que ce deuxième schéma de priorités permet également dans un *environnement dynamique*[20] une progression globale des messages. Cependant, il n'assure pas une progression individuelle de chaque message. Dans un tel environnement dynamique, les deux schémas de priorités précédents ne sont pas suffisants pour éliminer la famine de chaque message. Une extension possible, garantissant la délivrance à destination de chaque message, pour le premier schéma doit pouvoir :

1. assigner une priorité *unique* et *distincte* à chaque message $m$ en transit dans le réseau,
2. une fois, une priorité $p$ assignée à un message $m$, le système de routage ne peut assigner des priorités plus *élevées* qu'à un nombre *fini, borné* de messages pendant la *durée de vie* du message $m$ dans le réseau.

La condition 1 maintient la *linéarité* de l'ordre des priorités. La condition 2 est équivalente à la propriété de *finitude* dans le cas statique.

D'où une extension possible du premier schéma de priorité, $p = (b, n)$ tel que :

$$(p_1 = (b_1, n_1)) > (p_2 = (b_2, n_2)) \iff (b_1 < b_2) \vee ((b_1 = b_2) \wedge (n_1 < n_2)) \quad (3)$$

où $b$ est la *date de naissance* du message dans la couche de routage.

20. C'est-à-dire un environnement qui ne suppose aucune limitation sur l'injection des messages dans la couche de routage.

De manière similaire, pour étendre le deuxième schéma de priorités, on constate que la famine éventuelle d'un message vient du fait que des messages *nouvellement* injectés ayant des distances plus *courtes* par rapport à leurs destinations vaincront des messages déjà dans le réseau ayant des distances plus *longues*. Cette situation peut être rectifiée en assignant des priorités plus *faibles* aux messages plus *jeunes*. D'où l'extension $p = (a, d)$ telle que :

$$(p_1 = (a_1, d_1)) > (p_2 = (a_2, d_2)) \iff (a_1 > a_2) \vee ((a_1 = a_2) \wedge (d_1 < d_2)) \quad (4)$$

où $a$ est l'*âge du message*, i.e. le nombre de cycles de routage écoulés depuis l'injection du message dans la couche de routage. Finalement, on observe que si l'on redéfinit le schéma de priorités (3) en termes d'âge des messages, $p = (a, n)$ tel que :

$$(p_1 = (a_1, n_1)) > (p_2 = (a_2, n_2)) \iff (a_1 > a_2) \vee ((a_1 = a_2) \wedge (n_1 < n_2)) \quad (5)$$

les deux conditions précédemment citées restent satisfaites. La différence par rapport au schéma (3) est que, dans le schéma (5), la priorité affectée à un message est mise à jour après chaque cycle de routage.

Les auteurs montrent alors que les schémas d'ordre de priorités définis par les équations (4) et (5) évitent la contention et assurent la délivrance à destination de *tout* message en circulation dans le réseau, dans un environnement dynamique.

Les auteurs décrivent également une méthode pour assurer l'injection des messages dans la couche de routage, appelée "*Contrôle de l'injection des messages*".

Au niveau d'un nœud du réseau, le problème de l'injection des messages dans la couche de routage survient lorsque tous les tampons de ce nœud alloués à cette couche sont pleins. En effet, dès qu'un message non destiné au nœud en question arrive sur un canal d'entrée, il est *prioritairement* stocké dans le tampon interne correspondant à ce canal. S'il arrive qu'en raison d'un *afflux* de messages intermédiaires au niveau de ce nœud, tous les tampons internes sont pleins, il ne peut injecter ses propres messages. Ce nœud doit alors attendre qu'il ne reçoive plus de message sur un canal d'entrée donné et, qu'entre temps, le tampon interne correspondant à ce canal se vide, pour pouvoir injecter son message dans la couche de routage. La technique utilisée est un *protocole distribué* qui garantit l'occurrence d'un tel canal.

Par convention, lorsqu'un nœud ne possède pas de paquet utilisateur à injecter, il injectera un paquet vide "**NULL**"(qui ne nécessite donc aucun tampon de stockage). De ce fait, durant chaque *cycle de routage*, chaque nœud possède un paquet à injecter. Le protocole est basé sur une *synchronisation locale* entre chaque nœud et l'ensemble de ses voisins. De sorte qu'après chaque cycle de routage, le nombre de paquets injectés par chaque nœud du réseau *diffère* d'au plus

une *constante* fixée par rapport au nombre de paquets injectés par chacun de ses voisins. L'utilisation de paquets vides "NULL" prévient qu'un nœud n'ayant pas de paquets à injecter ne bloque ses nœuds voisins.

Chaque nœud du réseau maintient un *enregistrement* indiquant pour chaque nœud voisin, la *différence relative* entre le nombre de paquets injectés par ce voisin et le nœud en question. L'information nécessaire à la mise à jour périodique de cet enregistrement est supposée contenue dans l'en-tête de chaque paquet échangé avec un voisin, pour chaque cycle de routage. Cette information n'est autre que le nombre total de paquets injectés par un nœud durant le cycle de routage courant. Le protocole en détail est comme suit :

1. dès le démarrage du système, chaque nœud initialise l'enregistrement correspondant à chaque nœud voisin à 0,

2. au début d'un cycle de routage, chaque nœud examine la différence relative en nombre total d'injections de paquets enregistré pour chaque voisin et calcule leur minimum. Un nœud n'est autorisé à injecter un paquet en attente que si ce *minimum* est plus *grand* qu'une constante $(-k)$; i.e. ce nœud a injecté *moins* de $k$ paquets de plus que chacun de ses voisins,

3. si un nœud n'est pas autorisé à injecter son paquet en attente, il va à l'étape 5. S'il est autorisé, il examine son prochain paquet. Les paquets vides "NULL" sont toujours injectés par convention (puisqu'ils ne recquièrent aucun espace mémoire) ; alors que les paquets utilisateurs ne sont injectés que si le nœud trouve un *tampon vide* pour ce paquet,

4. chaque nœud calcule alors le nombre de paquets qu'il a *réellement* injecté durant le cycle courant, selon le résultat de l'étape 2 ; où l'injection des paquets vides est comptée comme une injection de paquets utilisateurs. Ce nombre est alors inséré dans l'en-tête de chaque paquet envoyé à un voisin,

5. à la fin du cycle de routage, chaque nœud met à jour l'enregistrement correspondant à chaque voisin, en utilisant les en-têtes de messages qu'il a reçu de ses voisins. En *rajoutant* le nombre d'injections que ce voisin a effectué pendant ce cycle de routage et en *soustrayant* son propre nombre d'injections. Le nœud est alors prêt pour le prochain cycle de routage en recommençant à l'étape 1.

Les auteurs montrent alors que la séquence d'actions ci-dessus garantit qu'après chaque cycle de routage, la différence entre le nombre total d'injections entre un nœud du réseau et chacun de ses voisins est au plus égale à $k$ (y compris l'injection de paquets vides).

Ce protocole distribué permet une différence *maximale* constante dans le temps, en nombre d'injections de paquets à travers les nœuds du réseau. Il établit une *discipline distribuée* pour que, à chaque fois qu'un nœud du réseau possède un message en attente d'injection et que le protocole le lui permet, ce nœud puisse coopérer afin d'assurer l'occurence d'un canal d'entrée non occupé, au bout d'un temps borné. Pour que cela soit possible, il faut que le nœud trouve donc un tampon vide au bout d'un temps fini. Les auteurs montrent alors qu'il est suffisant que les messages en circulation dans la couche de routage arrivent à destination et qu'ils soient consommés. C'est-à-dire que le système de routage garantisse l'absence de famine des messages circulant dans la couche de routage.

Finalement, les auteurs montrent que le schéma d'ordre de priorités défini aux équations (4) ou (5) avec la technique du "contrôle de l'injection des messages" définie ci-dessus assurent l'injection de tout message dans la couche de routage ainsi que sa délivrance à destination.

## 2.7 Conclusion

Dans ce chapitre, nous nous sommes intéressés aux problèmes du routage dans les architectures massivement parallèles, MIMD à mémoire distribuée.

Le noyau de routage assure l'une des fonctionnalités principales d'un système d'exploitation d'une architecture parallèle. Après avoir été longtemps défini de façon minimaliste, il commence à intégrer des critères de correction naguère ignorés et davantage de critères d'efficacité.

Par ailleurs, parmi les aspects étudiés dans ce chapitre, on peut noter les grandes tendances actuelles dans la conception et la réalisation de noyaux de routage :

1. l'utilisation du routage dynamique décentralisé,
2. l'utilisation de la technique de commutation "wormhole",
3. la prévention des dysfonctionnements,
4. l'intégration de la fonction de routage dans le silicium notamment en utilisant la technique du routage par intervalles [56].

Cette dernière tendance qui a pour aboutissement logique des "*processeurs de communication*" au niveau de chaque nœud du réseau est en passe d'être l'unique voie vers la puissance de traitement attendue des calculateurs parallèles. En effet, pour ne pas pénaliser les performances des "processeurs de calcul", qui ne cessent de croître, il s'avère nécessaire de "dégager" celui-ci de la fonction de gestion

des communications inter-processeurs. Cette fonction doit être assurée par un processeur spécifique, qui se charge de l'interface avec le réseau et permet ainsi un fonctionnement en régime "permanent" du processeur de calcul. C'est déjà le cas avec beaucoup de machines prototypes ou commerciales.

L'un des premiers circuits dédiés à la communication dans un réseau est le TORUS ROUTING CHIP [24], conçu pour un routage sans interblocage, avec un mécanisme de commutation de type "virtual cut-through" dans les tores. Un autre "processeur de communication" est le DIRECT ROUTING MODULE CHIP [42] qui constitue le circuit de communication de l'iPSC/2. Ce circuit utilise l'algorithme "e-cube" avec un mécanisme de commutation de circuit. Ces deux "processeurs de communication" ont été conçus pour des architectures à topologie bien définie. On s'oriente actuellement vers des processeurs de communication "*universels*", i.e. des circuits indépendants de la taille et de la topologie du réseau. Un des circuits s'inscrivant dans cette optique est le C104 d'INMOS [56]. C'est un circuit *programmable* permettant ainsi une communication par reconfiguration dynamique. Il intègre un *crossbar* $32 \times 32$ et utilise un algorithme de routage par intervalles avec une commutation de type "wormhole".

# Chapitre 3

# De l'antagonisme des critères de routage

Dans les chapitres précédents, nous avons fait ressortir le fait que l'une des problématiques principales des calculateurs parallèles à mémoire distribuée était la communication. Le gain en puissance de calcul escompté dans ce type de calculateur, en dépend directement.

Les performances des communications dépendent à la fois du médium et du logiciel de contrôle de ces communications. Le médium concerne principalement le réseau d'interconnexion. En effet, un *"bon"* réseau d'interconnexion est essentiel pour des performances de communication meilleures. On peut trouver une discussion sur les réseaux d'interconnexion adaptés aux calculateurs parallèles dans le rapport [75], selon différents points de vue. Parmi ces réseaux, le tore $3D$[1] représente un bon réseau.

Nous nous sommes intéressés au noyau de routage qui représente la composante de base du noyau de communication. L'objectif d'un tel noyau est : la *virtualisation du réseau d'interconnexion en un réseau complètement connecté.* Cette virtualisation du réseau est obtenue par le biais d'une **stratégie de routage**. Ainsi, l'on arrive au constat que le gain de performance est directement lié à la stratégie de routage. Une *"bonne"* stratégie est donc également essentielle. En effet, celle-ci constitue le "*squelette*" du noyau de routage sur lequel viennent se "*greffer*" ensuite l'ensemble des mécanismes de routage pour aboutir à un noyau de routage performant.

D'un autre côté, au regard de la spécification du problème de routage donnée dans le chapitre précédent, il apparaît que la majorité des critères à satisfaire sont relatifs à la stratégie de routage. Cette dernière apparaît donc comme un point clé dans la réalisation de noyaux de routage.

1. i.e. la grille rebouclée de dimension 3.

Les critères qu'une stratégie de routage idéale doit assurer sont :

1. la complétude,
2. l'évitement de l'interblocage,

Ce sont là les critères de correction. L'autre classe sont les critères d'efficacité :

1. le routage sur des chemins minimaux du réseau,
2. l'utilisation d'un espace mémoire indépendant de la taille du réseau en nombre de nœuds. Cet espace mémoire comprend deux composantes :
   (a) les tampons de stockage temporaire des messages utilisateurs,
   (b) l'espace mémoire de codage de l'information de routage,
3. la définition de plusieurs chemins de communication entre toute paire de nœuds,
4. l'utilisation de l'ensemble des liens de communication du réseau.

## 3.1 Corrélation entre les critères de routage

La stratégie de routage a été globalement caractérisée par la donnée de deux composantes :

1. une fonction de routage,
2. un mécanisme de virtualisation.

Pour illustrer cette caractérisation, considérons par exemple la méthode de *"Comptage de sauts"* [43]. Cette stratégie de routage est construite comme suit :

Les paquets arrivant sur un processeur sont répartis par classes, par exemple, en fonction de la distance parcourue depuis le nœud source. On construit alors un graphe orienté sans cycle sur l'ensemble des classes. Pour garantir l'absence d'interblocage, chaque fois qu'un paquet *traverse* un lien physique sa classe est incrémentée de 1. Le paquet est stocké au niveau du nœud récepteur, dans la classe juste supérieure par rapport à la classe où il se trouvait, au niveau du nœud émetteur.

Cette méthode nécessite donc un nombre de classes de tampons égal à la longueur du plus long chemin de communication plus 1. *Si le routage est effectué sur les plus courts chemins*, $D+1$ classes de tampons sont nécessaires dans chaque nœud, où $D$ est le diamètre du réseau.

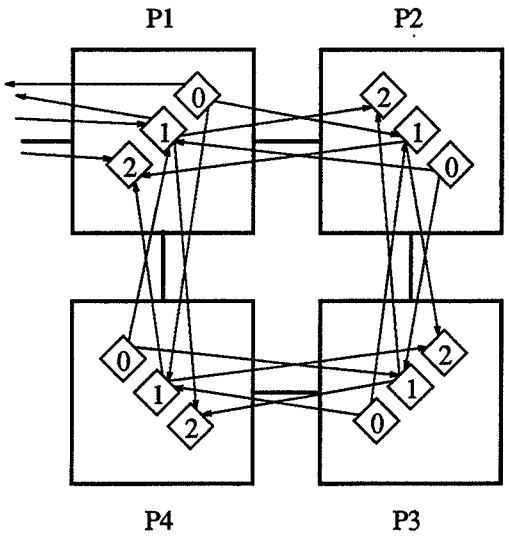


FIG. 3.1 – *Graphe de tampons pour la méthode de comptage de sauts*

Sur la figure 3.1, nous avons illustré cette méthode pour une grille $2 \times 2$, en supposant que les tampons de stockage sont communs à tous les liens de communication; i.e. ils sont associés aux nœuds.

On reconnaît donc bien une fonction de routage, en l'occurrence, un routage sur les plus courts chemins, et un mécanisme de virtualisation qui, dans le cas présent, peut être vu comme la réplication de chaque nœud en $D + 1$ nœuds virtuels, où chaque nœud virtuel possède sa propre classe de tampons.

La fonction de routage est l'élément qui, *potentiellement*, peut assurer les critères de routage énumérés précédemment. En effet, si pour un réseau donné cette fonction utilise tous les liens de communication du réseau, spécifie plusieurs voire tous les chemins minimaux pour tous les couples de nœuds *(source,destination)* du réseau, n'utilise pas dans cette spécification une forme de réplication des éléments du réseau et n'implique pas de cycle dans son graphe de dépendance[2], cette fonction de routage satisfait d'elle-même l'ensemble des critères que la stratégie de routage doit assurer.

Cependant, mis à part le cas de quelques réseaux d'interconnexion spécifiques, qui possèdent de bonnes propriétés topologiques, il est difficile qu'une fonction satisfasse "*de facto*", l'ensemble de ces critères, directement sur le réseau physique [89].

2. Pour l'évitement de l'interblocage.

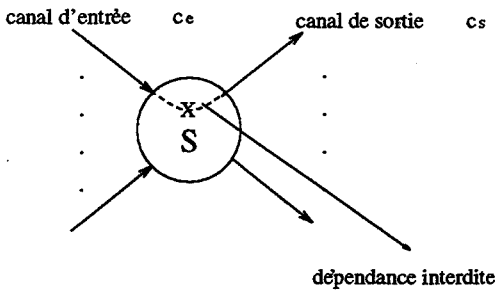


FIG. 3.2 – *Interblocage et élongation des chemins*

En effet, dans le cas général d'un réseau quelconque, la prévention de l'interblocage est un critère qui va à l'encontre d'un routage, *pour tous les couples de nœuds du réseau*, sur des chemins optimaux. La figure 3.2 illustre cet antagonisme. En effet, si une dépendance entre un canal d'entrée $c_e$ et un canal de sortie $c_s$ est à interdire, au niveau d'un sommet $s$, ceci implique que le canal $c_s$ ne peut jamais être successeur du canal $c_e$ dans un chemin de communication. En particulier, tous les plus courts chemins du réseau qui passent par ces deux canaux successivement, ne peuvent pas être empruntés consécutivement et sont en fait détournés.

Inversement, si le routage pour tous les couples de nœuds du réseau est à faire sur les plus courts chemins, cela induit des possibilités de cycles dans le graphe de dépendance, donc des interblocages potentiels.

Cet antagonisme est plus fort lorsqu'on considère un routage sur plusieurs, voire tous les chemins optimaux entre deux nœuds quelconques du réseau.

Les algorithmes de routage, **e-cube** pour l'hypercube et **XY** pour la grille, illustrent ce propos. Ces algorithmes satisfont tous les critères énoncés, sauf qu'ils ne permettent qu'un seul chemin de communication entre toute paire de nœuds (source, destination) du réseau. D'autant plus que les cas de l'hypercube et de la grille sont spécifiques. En effet, ces deux algorithmes exploitent les bonnes propriétés topologiques de ces réseaux d'interconnexion.

Un autre antagonisme existe également entre la longueur des chemins et l'espace mémoire nécessaire au codage de l'information de routage. En effet, on montre qu'il existe une corrélation directe entre la longueur des chemins de communication induits par une stratégie de routage sur un réseau, et la taille mémoire nécessaire au stockage de l'information de routage de cette stratégie [5, 33, 38, 78].

Pour intégrer un maximum de critères, la technique de la *virtualisation* est l'outil utilisé. On définit alors une réplication virtuelle d'une ou plusieurs res-

sources physiques de la machine, permettant par une gestion adéquate, d'obtenir ces critères. Cette réplication virtuelle pouvant être des réseaux virtuels définis sur le réseau physique, des canaux de communication virtuels qui partagent le même canal physique, des nœuds virtuels associés à un nœud physique, etc... .

Pour l'algorithme de routage dans l'hypercube, e-cube, cette notion de réplication virtuelle est utilisée par LI dans [67], à travers des canaux virtuels, pour justement permettre plusieurs chemins de communication entre toute paire de nœuds.

Au niveau physique, cette réplication virtuelle des ressources de la machine se traduit par davantage d'espace mémoire et un temps de gestion des ressources répliquées. Par exemple, lorsque ce sont des canaux virtuels qui sont utilisés, un canal physique est alors partagé en plusieurs canaux virtuels. Chaque canal virtuel admet son *propre* tampon mémoire et le temps de transfert des données sur le canal physique est partagé en tranches, une par canal virtuel. Mais alors, la question qui est directement induite est: *est-ce que la taille mémoire ainsi nécessaire, i.e. le nombre de tampons par canal physique*, n'est pas dépendante de la taille du réseau en nombre de nœuds?. Si cette taille en dépend, on se retrouve dans la situation où, pour résoudre un conflit, on en induit un autre.

Par conséquent, l'on arrive au constat que c'est l'ensemble des critères de routage, la minimisation de l'espace mémoire pour ses deux composantes, qui sont interdépendants et conflictuels. Pour la conception de stratégies de routage dans les architectures parallèles, cette interdépendance pose alors le problème de savoir quels critères doivent être priviligiés et par rapport à quel objectif.

## 3.2 Conception de stratégies de routage

Les méthodes de conception de stratégies de routage pour les réseaux de processeurs ne distinguent pas toujours nettement les deux composantes dégagées dans la section précédente. A savoir, une fonction de routage qui travaille directement sur le réseau physique et un mécanisme de virtualisation.

En effet, la technique de la virtualisation est directement utilisée dans la spécification de la fonction de routage du réseau physique [57, 46, 43, 80, 3, 45, 69]. Le problème de la conception d'une stratégie de routage est en effet traité sous un angle d'attaque d'un ensemble de critères, non exhaustif, en utilisant implicitement la technique de la virtualisation.

Au vu de la bibliographie étudiée dans le chapitre précédent, outre la propriété de complétude, parmi les critères ciblés dans la conception de stratégies de routage générales, i.e. des stratégies qui ne sont pas dédiées à un type de réseau

d'interconnexion, on peut distinguer globalement les trois points de vue suivants :

1. Des stratégies qui s'intéressent à la réduction de l'espace mémoire nécessaire au codage de l'information de routage [63, 6, 84, 64]. Le problème d'une telle approche survient lorsqu'on considère le critère du routage sur des chemins optimaux [89], ou encore la prévention de l'interblocage.

2. Des stratégies qui s'intéressent au problème de la panne physique. A notre point de vue, la reconfiguration dynamique du réseau [29, 9], lorsque cela est possible, est l'approche générale la plus connue.

3. Des stratégies qui s'attaquent prioritairement à la résolution de l'interblocage. A ce niveau, on peut distinguer deux grandes classes d'approches :

   (a) des méthodes basées sur la technique de la virtualisation [43, 40, 46, 39, 45, 3, 57]. Comme nous l'avons mentionné précédemment, de telles approches induisent un espace mémoire pour le stockage des messages, dépendant de la taille du réseau,

   (b) des méthodes basées sur l'existence d'un "réseau couvrant" du réseau d'interconnexion [80, 91, 73]. On entend ici par "réseau couvrant", un réseau connexe, contenant tous les nœuds du réseau original et où il existe un chemin de communication entre toutes les paires de nœuds. De telles structures de routage permettent un espace mémoire constant pour le stockage des messages utilisateurs. Cependant, le routage s'effectuant sur le réseau couvrant, il est ainsi difficile de satisfaire la contrainte d'optimalité des chemins de communication.

Ainsi, le problème du routage étant *fortement contraint*, lorsqu'une approche de résolution de ce problème doit tenir compte de l'ensemble des contraintes, il est difficile de concevoir une stratégie de routage correcte et efficace. Lorsqu'on s'intéresse à un sous-ensemble de critères, on arrive à obtenir des schémas de communication plus ou moins efficaces, mais qui ne satisfont pas la totalité des critères. Si l'on considère alors, les deux problèmes suivants :

1. *Existe-t-il un moyen d'améliorer ces schémas de communication pour intégrer davantage de critères et peut être même tous les critères?*,

2. *Pour concevoir une stratégie de routage, quels critères doivent être le point de départ dans cette conception, i.e. quels critères privilégiés?*

## 3.3 Proposition d'une méthode

Par une démarche méthodologique, nous proposons dans la seconde partie de ce rapport, la *communication multi-niveaux* et le *schéma de communication primaire* associé, pour la conception de stratégies de routage dans les réseaux de processeurs à mémoire distribuée [49, 47, 51].

Cette démarche est incrémentale. En effet, pour un réseau d'interconnexion donné, elle suggère de concevoir une stratégie de routage en deux étapes :

1. On part d'un schéma de communication donné de ce réseau, qui doit assurer au minimum les critères de correction et éventuellement des critères d'efficacité jusqu'à un certain degré; c'est le schéma de communication primaire **SCP**,

2. La méthode consiste ensuite à améliorer ce schéma, en un *schéma de communication général* **SCG**, qui intègre les critères d'efficacité. L'intégration de ces critères étant justement obtenue à l'aide de la technique de la virtualisation.

La communication multi-niveaux est basée sur une notion de **niveaux de communication**. Un niveau de communication est une *structure virtuelle de routage* induite par un ensemble de règles de routage sur ce réseau. Ces règles doivent satisfaire *nécessairement* les critères de correction et *éventuellement* des critères d'efficacité. Nous appelons cet ensemble de règles, le schéma de communication primaire *SCP*.

Le principe de la communication multi-niveaux, consiste à construire une *hiérarchie* de niveaux de communication associés au schéma primaire et ordonnés dans un ordre croissant, à partir du premier niveau jusqu'au dernier. Dans un même niveau $n$, le routage d'un message est effectué suivant le schéma de communication primaire et une transition du niveau $n$ au niveau $n+1$[3] est effectuée, chaque fois que l'on enfreint une règle du schéma primaire, dans la nécessité d'améliorer l'efficacité d'un chemin de communication *par rapport à un critère d'efficacité donné.*

Sur la figure 3.3, considérons un message $m$ à envoyer du nœud source $s$ au nœud destination $d$. Supposons que le chemin de communication spécifié par le schéma *SCP* soit le chemin en gros trait mentionné sur le réseau physique. Le routage du message $m$ commence initialement dans le réseau virtuel de niveau 1. Supposons qu'au niveau du nœud $i$, le critère d'efficacité spécifie le lien $l2$ comme lien de sortie pour l'envoi du message $m$ et non pas le lien $l1$. Pour satisfaire les critères de correction induits par le schéma primaire et le critère d'efficacité,

3. Juste supérieur.

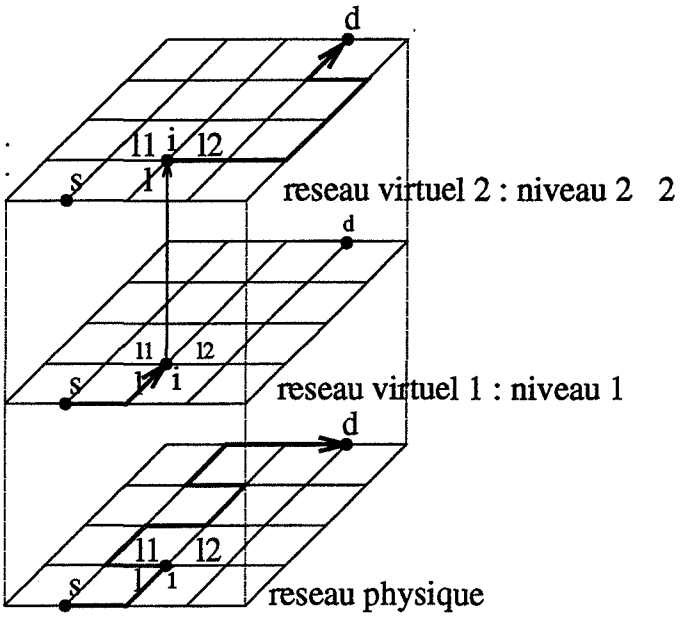


FIG. 3.3 – *Le principe de la communication multi-niveaux*

il suffit alors d'envoyer le message $m$ sur le lien $l2$ du réseau virtuel 2, et de continuer le routage du message $m$ dans ce réseau, suivant le schéma *SCP*.

Ceci représente le principe général de cette méthode, de manière abstraite. Aussi, on note que la hiérarchisation en niveaux de communication du schéma *SCP* cible un critère d'efficacité donné. Néanmoins, à travers les applications données dans la seconde partie, nous verrons que cette hiérarchisation permet de satisfaire d'autres critères d'efficacité jusqu'à un certain degré; lorsque le schéma de communication primaire est bien choisi.

En effet, l'incidence du schéma *SCP* sur le schéma général *SCG* obtenu à la fin est cruciale. Un facteur qui contribue à l'obtention d'un schéma *SCG* efficace est que le schéma primaire utilise tous les liens de communication du réseau d'interconnexion. La communication multi-niveaux n'étant qu'une méthode de virtualisation du schéma primaire, elle n'augmente donc pas les liens de communication utilisés par ce schéma.

Un autre facteur qui influe directement sur la qualité du schéma *SCG* est que le schéma *SCP* résolve *efficacement* le problème de l'interblocage. Le terme "efficace" regroupe deux aspects. D'une part, ce schéma ne doit pas utiliser un espace mémoire important, en particulier dépendant de la taille du réseau, pour le *stockage des messages utilisateurs*. Le meilleur cas est que cet espace soit d'*un*

*tampon par canal de communication.* D'autre part, ce schéma doit permettre le maximum de dépendances entre canaux de communication.

En partant d'une spécification formelle du problème du routage sans inteblocage, le schéma $SCP$ que nous proposons dans le chapitre 7 de la deuxième partie, tente de maximiser le nombre de dépendances à permettre, tout en assurant les deux propriétés mentionnés ci-dessus.

## 3.4 Interblocage et élongation des chemins

Le nombre de chemins de communication entre un couple de nœuds *(s,d)* est un critère relativisé par la longueur de ces chemins. En effet, lorsque la stratégie de routage satisfait à la propriété de complétude, si la contrainte sur la longueur de ces chemins n'est pas considérée, ce nombre est conceptuellement "*grand*"; puisque n'importe quel chemin d'origine $s$ et d'extrémité $d$ en est un. Par contre, si l'on contraint ce critère par la longueur des chemins, ou encore plus par des chemins optimaux, ce nombre est plus restreint.

De ce fait, parmi les critères d'efficacité que la hiérarchisation en niveaux de communication doit cibler, la longueur des chemins est le critère qui doit être pris en compte en priorité et ce, d'autant plus que ce critère influe directement sur les performances du noyau de routage. Conceptuellement, plus courts sont les chemins de communication, meilleurs seront les délais de communication.

D'ailleurs, ce critère est souvent utilisé comme une métrique de l'efficacité d'une stratégie de routage. En effet, dans [78], UPFAL et PELEG mesurent une telle efficacité par le facteur d'élongation des chemins d'une stratégie de routage[4]. Ce paramètre permet de donner une estimation du rallongement des chemins de communication.

Dans la seconde partie, nous nous intéresserons donc particulièrement à la *corrélation* entre la prévention de l'interblocage et le facteur d'élongation des chemins. Le but recherché est la conception de schémas $SCG$ qui soient sans interblocage et permettent en même temps, la *régulation* du facteur d'élongation des chemins en fonction du nombre de niveaux de communication.

4. Que nous avons déjà évoqué dans le chapitre précédent.

# Deuxième partie

# Une Méthode de Routage

# Chapitre 4

# Modèle et définitions

Concevoir ou prouver des algorithmes de routage n'a de sens que par rapport à un modèle précis, défini au préalable. Avec un modèle, les ambiguïtés sont levées et les preuves sont concises.

Nous considérons le modèle standard des architectures massivement parallèles à mémoire distribuée. En l'occurrence, un graphe orienté, symétrique, connexe et fini $G = (S, A)$, où :

- $S$ désigne l'ensemble des sommets[1], représentant les processeurs,
- $A$ désigne l'ensemble des arêtes, représentant les liens de communication.

Un *lien de communication* reliant un couple de processeurs $(p_1, p_2)$ est une voie de communication bidirectionnelle. Il est partagé par deux *canaux de communication*, qui représentent des voies de communication unidirectionnelles. Un premier canal relie le processeur $p_1$ au processeur $p_2$ et ne permet une communication que de $p_1$ à $p_2$. Le deuxième canal relie $p_2$ à $p_1$ et ne permet une communication que dans ce sens. Un canal de communication est modélisé par un *arc*. Ainsi, le graphe $G$ étant supposé orienté symétrique, on le représentera sous la forme d'un graphe non orienté en gardant en mémoire le fait qu'à chaque arête correspondent deux arcs opposés.

Un sommet ne peut communiquer directement qu'avec ses voisins. Les messages à transmettre entre un sommet *source* et un sommet *destination*, non adjacents, sont *routés* le long d'un *chemin de communication*.

Un message $m$ contient, dans l'ordre, une en-tête $h$ et des données utilisateur à transmettre. L'en-tête $h$ contient un ensemble d'informations de contrôle nécessaire au routage du message $m$, dont l'adresse de destination $d$.

1. Dans la suite, nous parlerons indifféremment de sommet ou de nœud.

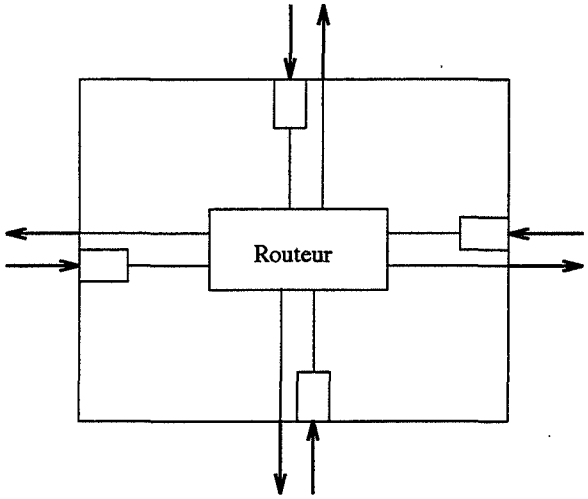


FIG. 4.1 – *Le modèle de nœud*

La figure 4.1 décrit le modèle de nœud que nous considérons vis-à-vis du processus de routage. Cette figure décrit un sommet avec quatre liens de communication. En particulier, *les tampons pour le stockage temporaire des messages utilisateurs sont associés aux canaux d'entrée.* Nous ne faisons pas d'hypothèse sur la taille exacte du tampon associé à un canal d'entrée. Ce paramètre dépend de la technique de commutation de données utilisée. Pour la suite de ce rapport, on peut supposer, sans perte de généralité, qu'à chaque canal d'entrée est associé *un seul tampon.*

Sur chaque nœud est implanté un *routeur*, logiciel ou matériel. Le rôle de ce routeur est d'acheminer les messages entre nœuds non voisins. Les routeurs réalisent de façon distribuée un *algorithme de routage* qui spécifie le chemin à suivre dans le réseau pour se rendre de tout nœud *source* à un nœud *destination.* L'algorithme de routage implante la *stratégie de routage.*

Localement, au niveau d'un nœud du réseau, un message $m$ muni d'une entête $h$ contenant l'adresse de destination $d$, qui arrive sur un canal d'entrée $c_e$, est stocké au niveau du tampon associé au canal $c_e$. Connaissant le canal d'entrée $c_e$ et la destination $d$ du message $m$, le routeur détermine alors un canal de sortie $c_s$ suivant lequel le message $m$ est ré-envoyé au prochain nœud dans son chemin vers la destination. La détermination du canal $c_s$ en fonction du canal $c_e$ et de la destination $d$, peut se faire par la consultation d'une table de routage locale.

Ainsi, le modèle mathématique que nous considérons pour la description de la stratégie de routage *au niveau d'un nœud du réseau*, est une *fonction* de la

forme :

$$f(c_e, d) = c_s$$

Sur la figure 4.1, nous n'avons modélisé que les tampons mémoire alloués à la *couche de routage*. En particulier, lorsqu'au niveau d'un nœud source $s$, un message $m$ est à injecter dans cette couche et à envoyer vers une destination $d$, on suppose que ce message est stocké au niveau d'un *tampon d'injection*[2]. Connaissant la destination $d$ du message $m$, le routeur détermine un canal de sortie $c_s$ réalisant un plus court chemin par rapport à cette destination. Le message $m$ est ensuite envoyé à partir du tampon d'injection suivant le canal $c_s$.

De même, lorsque le message $m$ arrive au nœud destination $d$ par un canal d'entrée $c_e$, le routeur l'envoie du tampon associé au canal $c_e$ vers un *tampon de sortie*[3], permettant sa consommation par les couches supérieures du noyau de communication.

Dans la suite de ce rapport, nous désignerons par les termes de, *schéma de communication* ou stratégie de routage, une méthode de routage sur un graphe $G$. Nous supposons qu'une telle méthode peut être modélisée par un ensemble de *règles de routage* $\{R_1, R_2, ..., R_r\}$ sur le graphe $G$. Les règles de routage $R_i$, $i = 1...r$, spécifient l'ensemble des dépendances $(c_e, c_s)$ permises (resp. interdites) dans le réseau de graphe $G$.

Aussi, nous donnons quelques définitions et notations et rappelons quelques notions de la théorie des graphes.

**Définition 4.1 : (facteur d'élongation)**

Pour une stratégie de routage $SR$ sur un graphe $G$, on définit le *facteur d'élongation* $\rho(SR, G)$ [78] par :

$$\rho(SR, G) = \max_{u,v \in S} \left\{ \frac{dist(SR,u,v)}{dist(u,v)} \right\}$$

où,

- $dist(SR, u, v)$ est la longueur d'un chemin minimal induit par $SR$, de $u$ à $v$,
- $dist(u, v)$ est la longueur d'un chemin de $u$ à $v$, dans le graphe $G$.

2. Qui n'est pas représenté sur la figure 4.1.
3. Qui n'est également pas représenté sur la figure 4.1.

**Définition 4.2 : (cycle, circuit)**

Dans un graphe $G$,

- un cycle est une suite ordonnée d'*arêtes* $(a_1, a_2, ..., a_q)$ toutes différentes, tel que chaque arête $a_i$ soit adjacente par l'une de ses extrémités à l'arête $a_{i-1 \ mod \ q}$ et par l'autre extrémité à l'arête $a_{i+1 \ mod \ q}$, pour $i = 1...q$,
- un circuit est une suite ordonnée d'*arcs* $(a_1, a_2, ..., a_q)$ tous différents, tel que chaque arc $a_i$ soit adjacent par son extrémité initiale à l'arc $a_{i-1 \ mod \ q}$ et par son extrémité finale à l'arc $a_{i+1 \ mod \ q}$, pour $i = 1...q$,

**Définition 4.3 : (cycle eulérien)**

Un cycle eulérien d'un graphe $G$ est un cycle passant une et une seule fois par chaque arête.

Evidemment tout graphe $G$ n'est pas eulérien[4]. Une condition nécessaire et suffisante pour qu'un graphe $G$ soit eulérien, est que tous ses sommets soient de degré pair [8]. Intuitivement, un cycle eulérien d'un graphe $G$ est un *parcours fermé* initié à partir d'un sommet *racine* $r$, qui passe par toutes les arêtes une et une seule fois.

Lorsque $G$ admet un tel parcours, nous notons :

**Notation 4.1 :**

- $(C, r)$ un cycle eulérien initié à partir de la racine $r$,
- $(C_+, r)$ le *circuit eulérien* initié à partir de la racine $r$, correspondant au cycle eulérien $(C, r)$, lorsque parcouru dans un sens,
- $(C_-, r)$ le circuit eulérien initié à partir de la même racine $r$, mais parcouru en sens inverse par rapport au circuit $(C_+, r)$,

Pour le graphe $G$, nous notons également :

**Notation 4.2 :** $|S|$ (resp. $|A|$), le cardinal de l'ensemble $S$ (resp. $A$), qui correspond au nombre de sommets (resp. arêtes) de $G$,

**Notation 4.3 :** $d(G)$, le diamètre du graphe $G$, i.e. la longueur du plus long, des plus courts chemins entre les paires de sommets de $G$ [8].

4. N'admet pas un cycle eulérien.

**Notation 4.4 :** $\delta$, le degré des sommets de $G$, lorsque celui-ci est régulier.

**Notation 4.5 :** Pour un sommet $s$, nous notons,

- $a_e$, un arc d'entrée de $s$, i.e. un arc dont l'extrémité finale est $s$, qui correspond à un canal d'entrée de $s$,
- $a_s$, un arc de sortie de $s$, i.e. un arc dont l'extrémité initiale est $s$, qui correspond à canal de sortie de $s$,
- $I_s$, l'ensemble des arcs incidents[5] à $s$,
- $V_s$, l'ensemble des sommets voisins de $s$.

5. Entrants ou sortants.

# Chapitre 5

# La communication multi-niveaux

## 5.1 Organisation du chapitre

Après avoir décrit le principe de la communication multi-niveaux appliqué à la paire *(absence d'interblocage, facteur d'élongation)*, nous présentons à la section 5.3 une formalisation, permettant de montrer que le schéma *SCG* obtenu à la fin est sans interblocage. Par rapport au modèle de routage que nous avons décrit dans le chapitre précédent, nous présentons ensuite à la section 5.4, la modélisation de la communication multi-niveaux. A la section 5.5, nous présentons deux applications de cette méthode aux grilles toriques multi-dimensionnelles, avec comme schéma *SCP*, le routage par cycle eulérien [72]. De même, dans la section 5.6, nous présentons une autre application aux grilles toriques de dimension quelconque, à base d'un schéma de communication primaire de type *e-cube.* Nous interprétons ensuite à la section 5.7, les résultats obtenus à la section 5.6. Enfin, à la section 5.8, nous faisons quelques remarques concernant le principe de cette méthode.

## 5.2 Le principe de la communication multi-niveaux

Considérons un réseau d'interconnexion $I$ de graphe $G$ et $SCP = \{R_j, j = 1 \ldots r\}$, un ensemble de *règles de routage sans interblocage* sur ce réseau: le *schéma de communication primaire.*

Pour prévenir l'interblocage, ces règles de routage interdisent un ensemble de dépendances entre canaux de communication, de telle sorte que le graphe de dépendance associé à la stratégie de routage, qu'elles induisent, soit acircuitique. Cet ensemble de dépendances interdites implique un certain *facteur d'élongation*

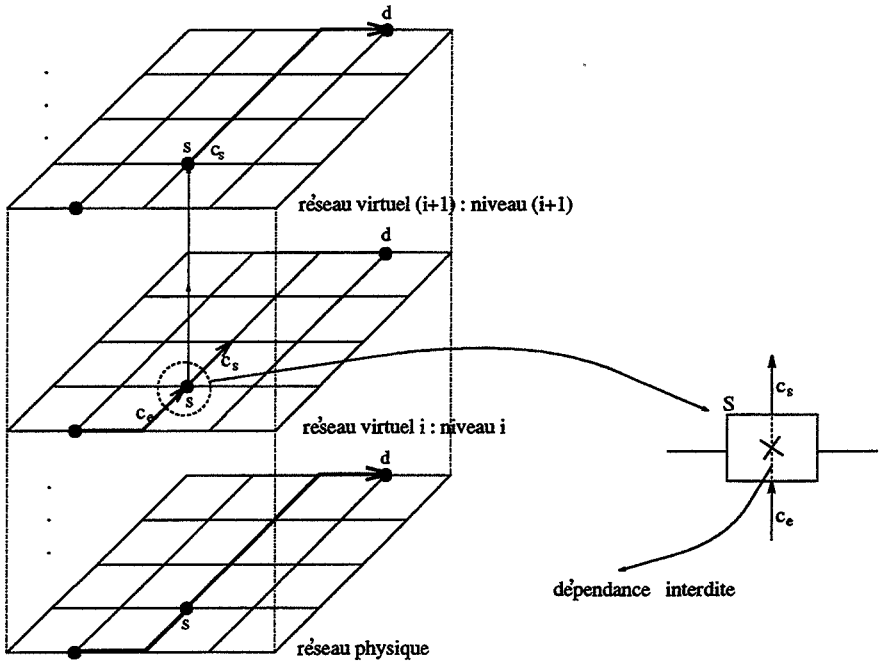


FIG. 5.1 – *Le principe de la communication multi-niveaux*

des chemins $\rho(SCP, G)$ du schéma $SCP$.

Supposons une *superposition* de $n$ *réseaux virtuels* $\{RV_j, j = 1 \ldots n\}$ identiques à $I$. $RV_j$ est le réseau associé au niveau de communication $j$. Le routage dans chaque réseau $RV_j$ est effectué selon le schéma de communication $SCP$. On se place dans le réseau virtuel $RV_i$ de niveau $i$. Supposons qu'au niveau d'un sommet $s$, un message $m$ arrivé par un canal d'entrée $c_e$ doit emprunter un canal de sortie $c_s$ qui définit le chemin minimal par rapport à sa destination $d$. Supposons également que la dépendance $(c_e, c_s)$ n'est pas permise par le schéma $SCP$, i.e. au niveau du sommet $s$, pour le canal d'entrée $c_e$ et la destination $d$, les règles de routage $\{R_j, j = 1 \ldots r\}$ ne spécifient pas $c_s$ comme canal de sortie. Le routage selon le principe de la communication multi-niveaux consiste alors à envoyer le message $m$ sur le canal $c_s$ du réseau $RV_{i+1}$, de niveau $i+1$, identique au canal $c_s$ du réseau $RV_i$ de niveau $i$, et à continuer le routage du message $m$ dans ce réseau suivant le schéma $SCP$ (voir figure 5.1).

On enfreint donc une règle de routage du schéma primaire, dans le but de router sur un chemin minimal, mais on monte d'un niveau de communication. Ainsi, par le fait de changer de niveau de communication, l'absence d'interblocage

est conservée[1] et aucune élongation de chemin n'a lieu au niveau du sommet $s$. Ce principe est appliqué au niveau de tous les sommets, chaque fois que l'on veut conserver le routage sur un chemin minimal et tant que l'on n'est pas au dernier niveau de communication. Au niveau du nœud source, le routage d'un message commence initialement dans le réseau de niveau 1.

Nous formalisons dans la section suivante, la notion de niveaux de communication et montrons que l'absence d'interblocage est conservée étant donné un schéma $SCP$ sans interblocage.

## 5.3 Propriété caractéristique de la communication multi-niveaux

Soient :
$G = (S, A)$, le graphe associé au réseau d'interconnexion $I$,
$N = \{1, 2, \ldots, n\}$, l'ensemble des niveaux de communication ($n \geq 2$),
$\Re = \{R_1, R_2, \ldots, R_r\}$, l'ensemble des règles de routage sans interblocage associées au schéma $SCP$.

Au niveau d'un sommet $s$ du graphe $G$, nous étendons le modèle mathématique pour la description de la stratégie de routage, avec la notion de niveau de communication, de la manière suivante :

**Définition 5.1 :**

$$\begin{aligned} F_s : A \times N \times S &\longrightarrow \wp(A \times N) \\ (a_e, n_c, d) &\longmapsto (a_s, n_s) \end{aligned}$$

la fonction $F_s$ donne, pour un arc d'entrée $a_e$ d'un message $m$, stocké dans le niveau courant $n_c$ et à destination de $d$, l'ensemble de tous les couples $(a_s, n_s)$ formés d'arcs de sortie avec leurs niveaux de stockage associés.

On définit la stratégie de routage sur l'ensemble du graphe $G = (S, A)$ par :

**Définition 5.2 :**

$$F = \bigcup_{s \in S} F_s$$

Au niveau du sommet $s$, la fonction $F_s$, est calculée d'après les deux règles `principales` suivantes :

- `Si` (l'une des règles $\{R_1, \ldots, R_r\}$ est utilisée pour le routage d'un message $m$) `Alors` $n_s = n_c$,

1. Nous montrons cela dans le paragraphe suivant.

• Si (on enfreint l'une des règles pour le routage de $m$ ) Alors $n_s = n_c + 1$.

**Remarque 5.1 :**
L'application $F_s$ peut être décomposée en :

$$F_s = F_s^1 \cup F_s^{1\succ 2} \cup F_s^2 \cup \cdots \cup F_s^{(n-1)} \cup F_s^{(n-1)\succ n} \cup F_s^n$$

tel que :

$$(a_s, n_s) = F_s^i((a_e, n_c, d)) \implies n_c = n_s = i \ \ \forall\, i, i = 1 \ldots n$$

c'est-à-dire la restriction de la fonction $F_s$ au niveau $i$ et,

$$(a_s, n_s) = F_s^{i\succ(i+1)}((a_e, n_c, d)) \implies n_c = i,\ n_s = i+1 \ \ \forall\, i, i = 1 \ldots (n-1)$$

qui correspond à la restriction de $F_s$ aux transitions du niveau $i$ au niveau $i+1$.

**Remarque 5.2 :**
De plus, les intersections deux à deux entre fonctions $F_s^i$ et/ou $F_s^{i\succ(i+1)}$ sont vides :

$$F_s^i \cap F_s^j = \emptyset \quad \forall i, j,\ \ i, j = 1 \ldots n \ \ i \neq j$$

et,

$$F_s^{i\succ(i+1)} \cap F_s^j = \emptyset \quad \forall\, i = 1 \ldots (n-1),\ j = 1 \ldots n$$

Pour caractériser l'interblocage avec le modèle ainsi défini, nous introduisons la notion de *graphe de dépendance.*

**Définition 5.3 :**
On dit qu'un couple $(a_e, n_c)$ dépend d'un couple $(a_s, n_s)$ si :

$$\exists\ d \in S, s \in S \ \ / \ \ F_s((a_e, n_c, d)) = (a_s, n_s),$$

et on note :

$$(a_e, n_c) \prec (a_s, n_s)$$

**Définition 5.4 :**
Le graphe de dépendance $D_{F_s}$, associé à la fonction $F_s$ est le graphe, tel que les sommets représentent les couples $(a_e, n_c)$ ou $(a_s, n_s)$ et tel que il existe un arc du sommet $(a_e, n_c)$ vers le sommet $(a_s, n_s)$, si $(a_e, n_c)$ dépend de $(a_s, n_s)$.

$$D_{F_s} = ((I_s, N), \{\ (a_e, n_c) \prec (a_s, n_s) \ \ / \ \ a_e, a_s \in I_s\ ,\ n_c, n_s \in N\ \})$$

le graphe de dépendance associé à la fonction $F$ est :

**Définition 5.5 :**

$$D_F = \bigcup_{s \in S} D_{F_s}$$

Nous avons alors la proposition suivante :

**Proposition 5.1 :** *Le routage selon le principe de la communication multi-niveaux sur tout ensemble de n niveaux ($n \geq 2$) associés à des règles de routage sans interblocage, est sans interblocage.*

**Preuve :** Pour montrer cette proposition, nous la montrons sur un ensemble de deux niveaux. La généralisation à un ensemble de $n$ niveaux peut être déduite directement. Nous montrons d'abord le lemme suivant :

**Lemme 5.1 :** *Un cycle dans $D_F$ correspond à une situation d'interblocage potentiel dans le réseau de graphe G et réciproquement.*

**Preuve ($\Leftarrow$) :** Supposons une situation d'interblocage entre un ensemble $\{s_i, i = 1 \ldots l\}$ de sommets. Il existe donc un cycle de demandes de requêtes entre ces $l$ sommets; chaque sommet $s_i$ contient un message $m_i$ stocké dans le niveau $n_i$, et ne peut l'envoyer à son voisin $s_{i+1}$ (les opérations arithmétiques sont *modulo* $l$). Soit $a_{e_i}$, respectivement $n_{c_i}$, l'arc par lequel $m_i$ est arrivé à $s_i$, respectivement le niveau où il est stocké. Soient $a_{s_i}$ (resp. $n_{s_i}$), l'arc de sortie (resp. le niveau de sortie) que $s_i$ a calculé pour envoyer le message $m_i$ à destination. Nous avons au niveau du sommet $s_i$ l'égalité suivante :

$$F_{s_i}((a_{e_i}, n_{c_i}, d_i)) = (a_{s_i}, n_{s_i})$$

$s_i$ ne peut envoyer son message $m_i$ dans trois cas :

1. $n_{c_i} = 1, n_{s_i} = 1$ et $s_{(i+1)}$ ne peut recevoir le message $m_i$ (son tampon de niveau 1, correspondant à l'arc d'entrée $a_{s_i}$, est plein),
2. $n_{c_i} = 1$, $n_{s_i} = 2$ et $s_{(i+1)}$ ne peut recevoir le message $m_i$ (son tampon de niveau 2, correspondant à l'arc d'entrée $a_{s_i}$, est plein),
3. $n_{c_i} = 2, n_{s_i} = 2$ et $s_{(i+1)}$ ne peut recevoir le message $m_i$ (pour la même raison qu'en 2).

- Supposons le dernier cas : au niveau de $s_i$, le chemin que $m_i$ doit emprunter est :

  $$F_{s_i}((a_{e_i}, 2, d_i)) = (a_{s_i}, 2)$$

  de même au niveau de $s_{i+1}$, $m_{i+1}$ doit emprunter le chemin,

  $$F_{s_{(i+1)}}((a_{s_i}, 2, d_{i+1})) = (a_{s_{i+1}}, 2)$$

et ne peut être envoyé à $s_{i+2}$.
Nous avons donc une suite de portions de chemins de la forme:

$$F_{s_i}((a_{e_i}, 2, d_i)) = (a_{s_i}, 2)$$

qui reboucle sur elle même, i.e.:

$$a_{s_i} = a_{e_{(i+1)}}$$

et tous les tampons de niveau 2 correspondant aux arcs d'entrée $a_{e_i}$, $i = 1 \ldots l$ sont pleins. Au niveau de chaque sommet $s_i$ nous avons une dépendance de la forme

$$(a_{e_i}, 2) \prec (a_{s_i}, 2)$$

cet ensemble de dépendances boucle sur lui même puisque:

$$a_{s_i} = a_{e_{i+1}}$$

et induit donc un cycle dans $D_F$.

- Supposons le deuxième cas: au niveau de $s_i$, $m_i$ doit emprunter le chemin:

$$F_{s_i}((a_{e_i}, 1, d_i)) = (a_{s_i}, 2),$$

il est donc stocké dans un tampon de niveau 1 et doit être envoyé à un tampon de niveau 2. Au niveau de $s_{i+1}$, le message $m_{i+1}$ est stocké dans le tampon de niveau 2, il ne peut donc emprunter qu'un chemin de la forme (en vertu des deux règles principales de routage):

$$F_{s_{i+1}}((a_{s_i}, 2, d_{i+1})) = (a_{s_{i+1}}, 2)$$

de même pour tous les autres messages $m_j, j > i + 1$, ils doivent emprunter un chemin de la forme:

$$F_{s_j}((a_{e_j}, 2, d_j)) = (a_{s_j}, 2)$$

Cependant au niveau de $s_{i-1}$, le message $m_{i-1}$ doit être envoyé au tampon de niveau 1 de $s_i$ pour avoir un cycle de demandes de requêtes, c'est-à-dire que $m_{i-1}$ doit emprunter le chemin:

$$F_{s_{i-1}}(a_{e_{i-1}}, 2, d_{i-1})) = (a_{e_i}, 1)$$

or cela contredit les règles de routage principales et cette situation ne peut avoir lieu, i.e. dans ce cas on ne peut avoir un cycle de demandes de requêtes, c'est-à-dire un interblocage.

- Supposons le premier cas : le raisonnement est similaire au cas 3 et nous avons bien une situation d'interblocage qui correspond à un cycle dans $D_F$.

En résumé, toute situation d'interblocage correspond à un cycle dans $D_F$.

($\Rightarrow$ informelle) : Supposons un cycle dans $D_F$, nous avons donc une série de dépendances qui bouclent sur elles-mêmes. Supposons que le tampon correspondant à chaque paire $(l, n)$ de cette boucle contient un message, ceci est une situation d'interblocage. Ceci achève la preuve du lemme 5.1. □

Pour montrer la proposition, il suffit de montrer que le graphe $D_F$ est acircuitique.

**Lemme 5.2:** *Le graphe de dépendance $D_F$ est acircuitique.*

**Preuve :** Pour un schéma de communication à deux niveaux, nous savons que :

$$F = \bigcup_{s \in S} F_s = \bigcup_{s \in S} (F_s^1 \cup F_s^{1 \succ 2} \cup F_s^2)$$

donc,

$$D_F = \bigcup_{s \in S} D_{F_s} = \bigcup_{s \in S} (D_{F_s^1} \cup D_{F_s^{1 \succ 2}} \cup D_{F_s^2})$$

ou encore,

$$D_F = (\bigcup_{s \in S} D_{F_s^1}) \bigcup (\bigcup_{s \in S} D_{F_s^{1 \succ 2}}) \bigcup (\bigcup_{s \in S} D_{F_s^2})$$

d'où :

$$D_F = D_{F^1} \cup D_{F^{1 \succ 2}} \cup D_{F^2}$$

$D_{F^1}$ correspond au graphe de dépendance de la fonction $F$ sur le niveau 1, $D_{F^2}$ pour le niveau 2 et $D_{F^{1 \succ 2}}$ correspond aux transitions de niveaux (1 vers 2). D'après la proposition 1, $D_{F^1}$ et $D_{F^2}$ ne contiennent pas de cycles, $D_{F^{1 \succ 2}}$ est tel que :

$$D_{F^{1 \succ 2}} = \{(a_e, 1) \prec (a_s, 2), \quad a_e, a_s \in A\}$$

nous avons vu précédemment que ce type de dépendances ne peut pas induire de cycle puisqu'on ne peut jamais refermer un chemin qui en contient au moins une. D'un autre côté, l'intersection deux à deux entre $F^1$, $F^2$ et $F^{1 \succ 2}$ étant vide, il en est de même pour $D_{F^1}$, $D_{F^2}$ et $D_{F^{1 \succ 2}}$, l'union de ces trois ensembles ne contient donc pas de cycle.

En conclusion, dans tous les cas il n'existe pas de cycle dans $D_F$. Ceci termine la preuve du lemme 5.2 et par conséquent la preuve de la proposition 5.1. □

En plus de la prévention de l'interblocage, le schéma de communication général $SCG$ qui vient d'être défini, améliore la qualité du facteur d'élongation des chemins, par rapport au schéma $SCP$. En effet, il permet de lever des restrictions induites par la communication sur un seul niveau et implique de ce fait un facteur d'élongation $\rho(SCG, G) < \rho(SCP, G)$.

Comme première estimation, on peut affirmer que la qualité du paramètre $\rho(SCG, G)$ dépend du nombre $n$ de niveaux de communication. De sorte que, ce nombre permet de contrôler la qualité de ce facteur. Conceptuellement, plus le nombre $n$ est grand, plus le facteur d'élongation $\rho(SCG, G)$ se rapproche de 1.

Pour un réseau d'interconnexion $I$ donné, la minimisation du nombre de niveaux pour avoir un facteur d'élongation unitaire est en fait le but ultime recherché et représente le point crucial de cette méthode.

Cependant, là encore, il existe une corrélation directe entre le schéma $SCP$, le réseau d'interconnexion $I$ et le nombre $n$ de niveaux[2]. En effet, il est évident que plus le schéma $SCP$ admet de bonnes propriétés de routage, moins le nombre de niveaux $n$ est grand. Par exemple, plus le schéma $SCP$ permet de dépendance entre canaux de communication, moins le nombre $n$ est grand. A cette première inter-dépendance entre le schéma $SCP$ et le nombre $n$, il faut ajouter les caractéristiques topologiques du réseau d'interconnexion. De sorte que pour un schéma $SCP$ donné, le nombre de niveaux de communication nécessaires pour un facteur d'élongation unitaire, peut être différent d'un réseau à un autre.

Ainsi, tout dépend de la manière dont on pose le problème et particulièrement de la donnée initiale de ce problème. En d'autres termes, les interprétations possibles sont :

1. Est-ce que la donnée initiale est le réseau d'interconnexion $I$, et le problème posé est le *routage efficace* dans ce réseau. A ce moment là, il faut trouver un "*meilleur*" schéma $SCP$ pour $I$, qui minimise le nombre $n$ de niveaux,

2. Ou, est ce que la donnée initiale est le schéma $SCP$, qui rappelons le doit être correct, et le problème posé est de *mesurer l'efficacité* de ce schéma. A ce moment là, une mesure possible de l'efficacité peut être le nombre de niveaux nécessaire pour un facteur d'élongation unitaire, pour différents réseaux,

2. Les résultats exposés dans ce chapitre tendent à le montrer.

3. Ou encore, est ce que la donnée initiale est le schéma général *SCG*, et le problème posé est de savoir si une telle approche pour l'intégration des critères d'efficacité est bonne ou pas?

En ce qui nous concerne, le but recherché est plutôt de répondre à la dernière question. Pour cela, nous illustrerons la méthode à travers deux schémas *SCP* différents: le routage par cycle eulérien [72] et le routage suivant l'algorithme e-cube. Dans les deux cas, nous appliquerons le schéma *SCG* résultant à un réseau particulier "*assez difficile*": la grille torique. Les résultats de ces applications permettront de valider l'efficacité de la méthodologie.

Pour le cas de la méthode de routage par cycle eulérien, nous donnons deux applications différentes. En effet, du fait que cette méthode utilise un cycle eulérien comme structure de routage sur le réseau, un premier cycle eulérien particulier de la grille torique induit un nombre de niveaux de communication pour un facteur d'élongation unitaire, dépendant de la dimension de la grille. Nous améliorons ensuite ce nombre en 2 niveaux, en utilisant un autre parcours eulérien de ce réseau.

## 5.4 Modélisation de la communication multi-niveaux

La figure 5.2 représente une modélisation de la communication multi-niveaux, étendant le modèle de nœud de la figure 4.1, que nous avons considéré dans le chapitre précédent. Dans ce nouveau modèle, deux extensions principales sont apportées:

- à chaque arc d'entrée sont associés plusieurs tampons: *un tampon par niveau de communication*,
- un processus *"aiguilleur" (PA)*, qui fait partie intégrante du routeur, est également associé à chaque arc d'entrée. Ce processus permet d'*aiguiller* un message reçu sur un arc d'entrée, vers son tampon de stockage adéquat.

Par rapport au modèle d'exécution du processus de routage, que nous avons décrit au chapitre précédent, on peut à présent enrichir ce modèle comme suit:

*L'en-tête h d'un message contient également le niveau n de stockage du message. Localement, au niveau d'un sommet s, un message m qui arrive sur un canal d'entrée $c_e$ est immédiatement réceptionné par le processus PA, qui extrait de l'en-tête h le niveau $n_c$ où le message m doit être stocké et aiguille ce dernier vers le tampon qui correspond à ce niveau. Connaissant, le niveau courant $n_c$, la*

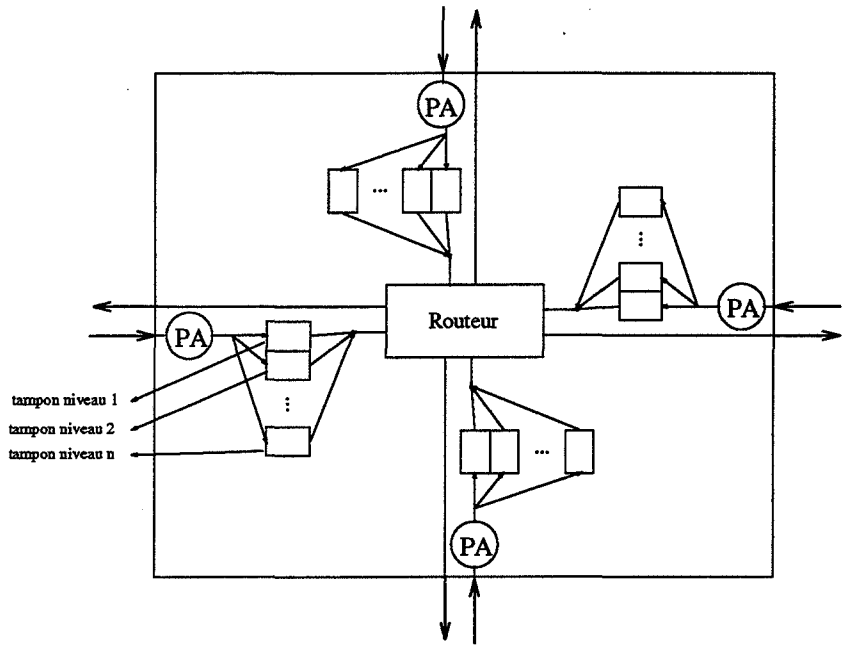


FIG. 5.2 – *Le modèle de nœud pour la communication multi-niveaux*

*destination d et le canal d'entrée $c_e$ du message m, le routeur détermine alors, un canal de sortie $c_s$ et un niveau de sortie $n_s$ du message m. Le nouveau niveau $n_s$ est alors inséré dans l'en-tête h et le message m est envoyé suivant le canal $c_s$. La détermination du canal $c_s$ et du niveau $n_s$ peut se faire par la consultation d'une table de routage locale.*

*Au niveau d'un nœud source s, si un message m est à injecter dans la couche de routage et à envoyer vers une destination d, le routeur détermine un canal de sortie $c_s$ réalisant un plus court chemin par rapport à la destination d, il insère la valeur* 1 *dans le champ niveau de l'en-tête h du message m et envoie le message m suivant le canal $c_s$.*

La modélisation de la communication multi-niveaux ci-dessus représente en fait la technique d'implantation de cette méthode de routage. En l'occurrence, au niveau d'un nœud du réseau, un niveau de communication correspond à un *tampon* de stockage par canal d'entrée et la hiérarchisation en plusieurs niveaux correspond à une duplication de ce tampon en autant de niveaux de communication, par canal d'entrée.

Lorsqu'un message est envoyé d'un processeur $p_i$ à un processeur $p_{i+1}$ sur le même niveau, il est stocké (au niveau de $p_{i+1}$) dans le tampon de même niveau

que le tampon du processeur émetteur. Lorsque le message doit monter d'un niveau de communication, il est stocké dans le tampon de niveau juste supérieur au niveau du processeur récepteur.

Ainsi, nous retrouvons bien le constat que nous avons fait dans la première partie de ce rapport. A savoir que l'intégration de plus de critères de routage est obtenue à travers le principe de virtualisation. En effet, dans notre cas, on virtualise la ressource tampon. Ce qui est tout à fait équivalent à dire que l'on virtualise la ressource canal physique en plusieurs canaux virtuels, tels que chaque canal virtuel admette son propre tampon. Tout le problème est la taille des espaces mémoire induits; i.e. le nombre de tampons par canal d'entrée.

En d'autres termes, ceci est équivalent à poser le problème de l'efficacité d'une telle méthode de virtualisation.

## 5.5 Application 1 : la communication multi-niveaux selon les règles de routage par cycle eulérien

La communication multi-niveaux a été implantée sur un réseau de TRANSPUTERS. Nous détaillerons au chapitre suivant cette implantation. Nous avons choisi le routage par cycle eulérien [72] comme schéma de communication primaire. Ce schéma utilise un cycle eulérien [8] du graphe associé au réseau d'interconnexion des processeurs comme structure de routage; lorsque ce graphe en admet un. Pour des réseaux de TRANSPUTERS, le graphe associé admet toujours un tel cycle, puisque le TRANSPUTER est un processeur à 4 liens de communication. Tous les sommets du graphe associé sont donc de degré pair.

La méthode de routage par cycle eulérien induit un espace mémoire, pour le stockage temporaire des messages utilisateurs, indépendant de la taille du réseau. En l'occurrence, un tampon par canal de communication. Cette méthode utilise tous les liens de communication, cependant elle n'assure pas, pour tous les couples de nœuds *(source,destination)*, un routage sur des chemins minimaux.

Il est également important de noter qu'en dépit des intéressantes propriétés de routage de cette méthode, celles-ci sont très liées à la qualité du cycle eulérien utilisé. En particulier, le parcours eulérien admet une incidence directe sur la qualité du facteur d'élongation des chemins. Le calcul d'un "meilleur" cycle est prohibitif et non envisageable, de sorte que la communication multi-niveaux s'avère être une réponse intéressante à cette incidence, particulièrement dans le cas de la grille torique. Nous rappelons d'abord le principe de cette méthode.

### 5.5.1 La méthode de routage par cycle eulérien

Soit $G = (S, A)$, le graphe d'un réseau d'interconnexion $I$. Soit $(C, r)$ un cycle eulérien[3] de $G$. Au couple $(C, r)$, nous associons le graphe défini comme suit :

**Définition 5.6 :**
Soit $G' = (S', A')$ un anneau associé au cycle eulérien $(C, r)$, tel que $|S'| = |A|$ et $|A'| = |A|$.
Intuitivement, le graphe $G'$ consiste en une *"ouverture"* du cycle eulérien $(C, r)$. Soit $f$ une numérotation des nœuds de $G'$ définie comme suit :

**Définition 5.7 :**

$$\begin{aligned} f : S' &\longrightarrow IN \times IN \\ s &\longmapsto (f_1(s), f_2(s)) \end{aligned}$$

tel que :

- $f_1(r) = 1$,
- $f_1(s)$ représente le *rang* du sommet $s$ dans $G'$. On suppose que les sommets de $G'$ sont numérotés incrémentalement à partir de 1 lors du parcours eulérien $(C, r)$ de $G$,
- $f_2(s)$ représente l'identité du sommet de $G$ dont $s$ est une occurrence.

De même, au couple $(C_+, r)$ (resp. $(C_-, r)$), nous associons le graphe correspondant suivant :

**Définition 5.8 :**
Soit $G'_+$ (resp. $G'_-$), le graphe orienté, obtenu à partir de $G'$ lorsqu'on parcourt, partant de $r$, ses nœuds dans l'ordre croissant (resp. décroissant) par rapport à la fonction $f_1$.
Considérons les définitions suivantes :

**Définition 5.9 :**

- un arc de $G'_+$ (resp. $G'_-$) est dit direct (resp. indirect),
- un arc de $G$ est dit direct (resp. indirect), si l'arc correspondant de $G'$ est direct (resp. indirect),

3. On suppose que $G$ en admet un.

- tout arc de $G'_+$ (resp. $G'_-$) est numéroté par le numéro associé (selon la fonction $f_1$) à son extrémité initiale,
- tout arc de $G$ est numéroté par le numéro de l'arc correspondant de $G'$,

Pour prévenir l'interblocage, la méthode de routage par cycle eulérien utilise la caractérisation de DALLY et SEITZ. Elle consiste à router dans le réseau de graphe $G$ en respectant les règles de dépendance suivantes :

- un arc direct ne peut dépendre d'un arc indirect,
- un arc direct ne peut dépendre d'un arc direct de numéro inférieur,
- un arc indirect ne peut dépendre d'un arc indirect de numéro supérieur.

On montre [72], que ces règles routent sans interblocage dans le réseau de graphe $G$.

Sur la base des résultats de simulation, des applications aux *grilles toriques multi-dimensionnelles* ont été analysées. Pour un parcours particulier, il en est ressorti que lorsque le nombre de niveaux est égal à la dimension de la grille, la communication multi-niveaux basée sur le routage par cycle eulérien route sur des chemins optimaux. Dans le paragraphe qui suit, nous décrivons le parcours en question et montrons le résultat.

### 5.5.2 Application à la grille torique

La grille torique (ou tore) de dimension $d$ et de base $k$ peut être vue comme une grille de dimension $d$ et de base $k$, avec $k$ nœuds interconnectés en un cycle pour chaque dimension[4]. Formellement, ce réseau est défini comme le produit cartésien de $d$ cycles de $k$ nœuds[5] [25].

La figure 5.3 décrit un tore de dimension 2 et de base 5. La figure 5.4 décrit un tore de dimension 3 et de base 3. Sur cette figure, nous avons représenté les trois 3-cycles dont le produit cartésien donne le tore représenté.

Un tore de dimension $d$ et de base $k$ possède $k^d$ nœuds et $d * k^d$ liens. C'est un graphe de CAYLEY symétrique, de degré $2d$ et de diamètre $d\lfloor k/2 \rfloor$ [25]. Une grille torique de dimension $d$ admet toujours un parcours eulérien, puisque tous ses nœuds sont de degré pair. Une autre propriété intéressante de ce réseau est qu'il est *hamiltonien*[6][25].

4. Dans la suite nous utiliserons la notation tore$(d, k)$ pour désigner un tel réseau.
5. Dans la suite, nous parlerons de $k$-cycle
6. Il admet un parcours hamiltonien.

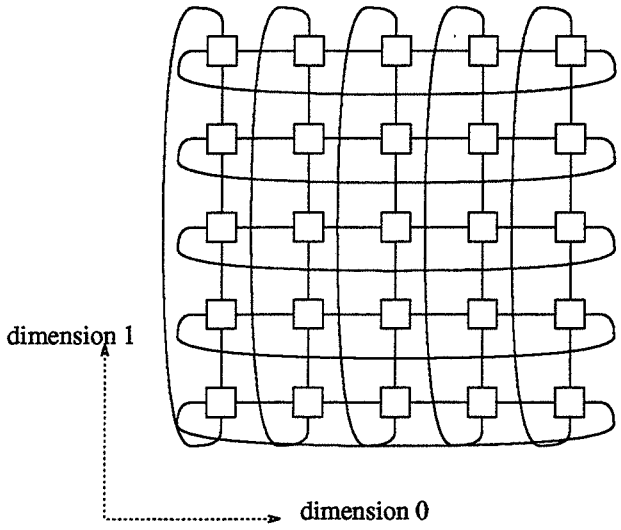


FIG. 5.3 – *Une grille torique de dimension* 2 *et de base* 5

Ce type de réseau d'interconnexion a été utilisé dans plusieurs calculateurs massivement parallèles [22, 69]. Il possède de bonnes propriétés topologiques [81]. Cependant, vis-à-vis du routage, ce n'est que ces dernières années que l'on assiste à l'emergence d'algorithmes de routage efficaces pour ce type de réseaux. Par rapport à la grille, le problème sont les liens de rebouclage, qui impliquent qu'un routage de type $e-cube$ induit des possibilités d'interblocage.

Le tableau 5.5 donne quelques résultats importants du routage dans le tore. En particulier dans [21], CYPHER et GRAVANO présentent un algorithme sans interblocage, adaptatif qui route sur des chemins optimaux et qui utilise deux liens virtuels par lien physique, donc deux niveaux de communication.

Cependant, cet algorithme prévient l'interblocage en se basant sur une caractérisation similaire à celle décrite dans [28]. A savoir qu'il induit des cycles dans le graphe de dépendance de sa fonction de routage. Cet algorithme n'est donc pas tolérant aux congestions locales, puisque pour éviter l'interblocage, certains chemins de communication entre couples de nœuds $(source, destination)$ ne sont pas permis[7].

Pour la représentation mémoire de la fonction de routage, les auteurs donnent deux représentations différentes. Une première représentation sous forme de fonctions arithmétiques nécessite un espace mémoire constant. Cependant ces fonc-

7. Comme nous l'avons explicité dans la section sur l'interblocage, du chapitre 2 de la première partie.

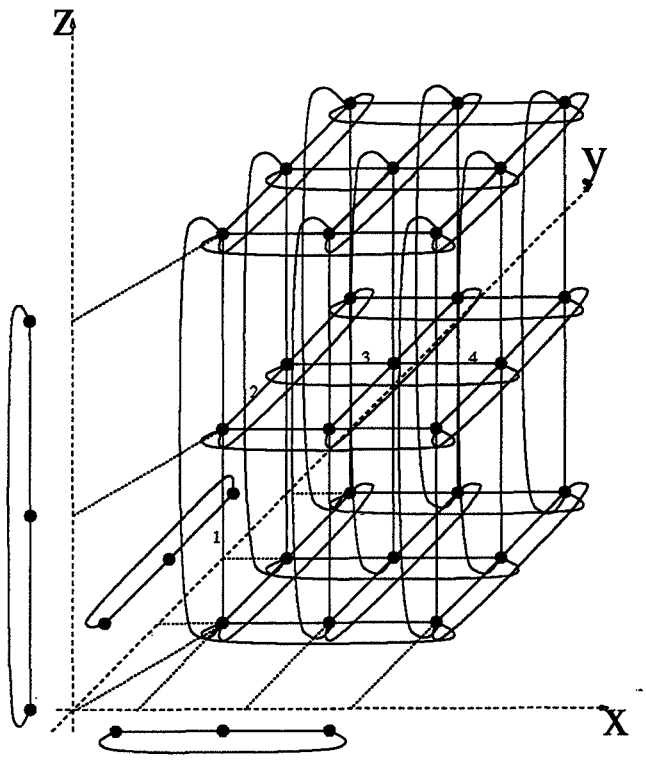


FIG. 5.4 – *Une grille torique de dimension* 3 *et de base* 3

| Algo | Sans Interblocage | Plus Court Chemin | Adaptativite' | Espace Me'moire | |
|---|---|---|---|---|---|
| | | | | Informations de routage | Stockage des Messages |
| e-cube | Non | Oui | Non | o(1) | o(1) |
| Dally & Seitz 86 | Oui | Non | Non | o(1) | 2 cv par cp |
| Linder & Harden 91 | Oui | Oui | Oui | o(1) | $(n+1)\,2^{n-2}$ |
| Gravano & Cypher 94 | Oui | Oui | Oui | $o(k^n)$ | 2 cv par cp |
| Hadim & Sakho 95 | Oui | Oui | Non | o(1) | (n+1) |

FIG. 5.5 – *Quelques résultats de routage dans la grille torique (cv : canal virtuel, cp : canal physique)*

tions sont recursives et requièrent un temps de calcul qui n'est pas constant, il dépend des paramètres $d$ et $k$. La deuxième représentation est une tabulation des fonctions précédentes et nécessite *deux* tables nécessitant chacune un espace mémoire en $O(k^d)$.

L'algorithme qui résulte de l'application de la communication multi-niveaux basée sur le routage par cycle eulérien, décrit dans ce paragraphe, est sans interblocage et tel que son graphe de dépendance est acircuitique, adaptatif, route sur des chemins optimaux et utilise $d$ niveaux de communication, donc $d$ canaux virtuels par canal physique, et une *seule* table de routage de taille $O(k^d)$.

#### 5.5.2.1 Un algorithme de routage dans les grilles toriques

Dans un système d'axes orthogonaux de cardinal $d$, tout nœud $x$ d'un tore$(d,k)$ peut être représenté par $d$ coordonnées $(x_0, x_1, ..., x_{d-1})$, où $x_i$ est la coordonnée suivant la dimension $i$, pour $i = 0, ..., d-1$ et $x_i = 0, ..., k-1$. $x$ est connecté à tous les nœuds qui diffèrent sur une seule coordonnée par $(\pm 1 \ mod \ k)$ et se trouve à l'intersection de $d$ $k$-cycles (voir figures 5.3, 5.4).

Suivant ce système de représentation du tore(d,k), considérons le parcours eulérien de ce réseau défini par la procédure *récursive* suivante :

```
Procedure Cycle_Euler_Tore(d,k,(0,...,(d-1)))
Debut
    index=0;
    Tantque (index<(k-1))
       Faire
          Si (d>1) Alors
                     Aller du noeud (0,...,0,index)
                                  au noeud (0,...,0,index+1);
                     Cycle_Euler_Tore((d-1),k,(0,...,(d-2)));
                  Sinon
                     Boucler le k-cycle suivant la dimension 0
                     dans l'ordre croissant de la coordonne'e;
                     index=k+1;
          Fsi
          index=index+1;
       Fait
    Si (d>1) Alors
               Aller du noeud (0,...,0,(k-1))
                     au noeud (0,...,0);
               Cycle_Euler_Tore*((d-1),k,(0,...,(d-2)));
    Fsi
Fin
```

Cette procédure effectue un parcours eulérien du tore(d,k) (d et k, étant les deux premiers paramètres d'entrée de la procédure) représenté dans un système d'axes dont les dimensions sont $0, ..., (d-1)$ (le dernier paramètre d'entrée) (voir figure 5.6).

Partant du nœud de coordonnées $(0, ..., x_{d-1} = 0)$, on passe au nœud de coordonnées $(0, ., 0, x_{d-1} = 1)$ suivant la dernière dimension $d-1$. A partir de ce dernier nœud, on effectue récursivement un parcours eulérien du tore(d-1,k) dont tous les nœuds ont pour coordonnée $x_{d-1} = 1$. Ce parcours s'achève au nœud de coordonnées $(0, ., 0, x_{d-1} = 1)$. On passe alors au nœud $(0, ., 0, x_{d-1} = 2)$. De même, à partir de ce dernier, on effectue récursivement un parcours eulérien du tore(d-1,k) dont tous les nœuds ont pour coordonnée $x_{d-1} = 2$ et ainsi de suite. Lorsque le parcours du tore(d-1,k) dont la coordonnée $x_{d-1} = k-1$ aura été effectué, on passe alors du nœud de coordonnées $(0, ., 0, x_{d-1} = k-1)$ au nœud de coordonnées $(0, ., 0, x_{d-1} = 0)$, suivant le lien de rebouclage. On effectue ensuite récursivement le parcours eulérien du tore(d-1,k) dont la coordonnée $x_{d-1} = 0$ avec l'hypothèse (qui correspond à (*) dans l'appel récursif de la procédure):

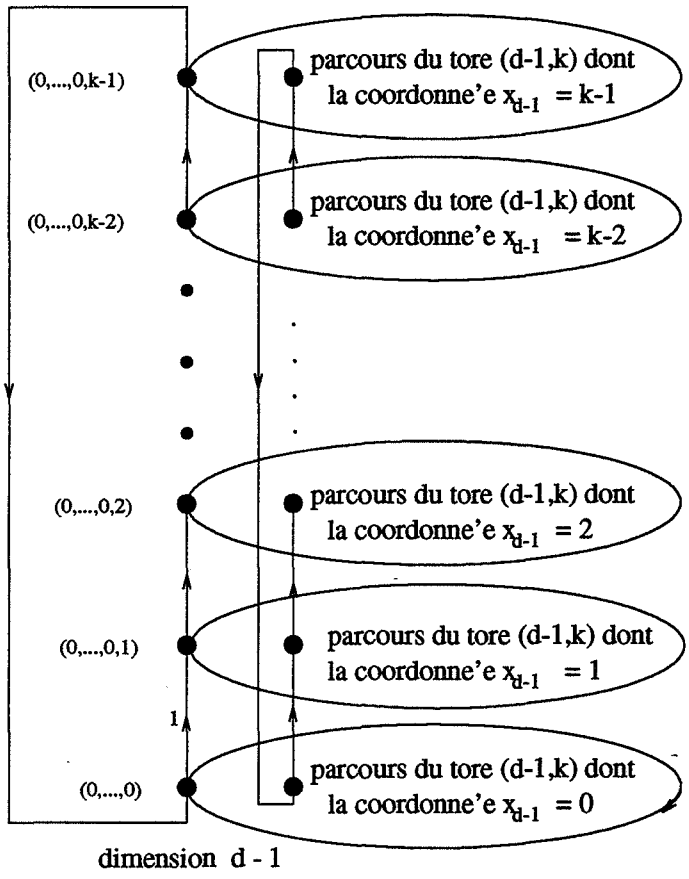


FIG. 5.6 – *Le parcours eulérien dans un tore de dimension d et de base k*

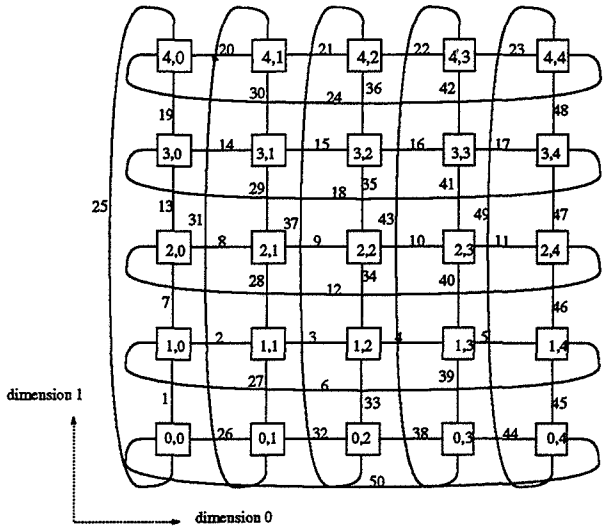


FIG. 5.7 – *Le parcours eulérien dans un tore(2,5)*

*arrivé au niveau d'un nœud quelconque de ce tore(d-1,k), on commence d'abord par boucler le $k-$ cycle suivant la dimension $d-1$ dans l'ordre croissant de la coordonnée $x_{d-1}$.*

La figure 5.7 illustre ce parcours eulérien particulier dans le cas d'un tore de dimension 2 et de base 5. Nous avons représenté le numérotage des arcs directs suivant la fonction $f_1$, définie au paragraphe 5.5.1. La figure 5.8 illustre le parcours pour une grille torique de dimension 3 et de base 3.

Ce parcours eulérien particulier du tore$(d,k)$ permet d'énoncer le résultat suivant :

**Théorème 5.1 :** *Le routage dans un tore(d,k), selon le principe de la communication multi-niveaux basée sur les règles de routage par cycle eulérien et utilisant le cycle défini précédemment, route sur des chemins minimaux avec d niveaux de communication,* $\forall\ d \geq 2,\ k \geq 1$.

**Preuve :** Soit $(C,r)$ un cycle eulérien du tore$(d,k)$, à partir du nœud racine r de coordonnée $(0,...,0)$, suivant la procédure définie précédemment.

Nous identifions le circuit $(C_+,r)$ au sens de parcours défini par la procédure, i.e. le sens direct. $(C_-,r)$ est alors le circuit parcouru dans le sens inverse. Les arcs (directs et indirects) de ce tore sont numérotés à l'issue de ce parcours, suivant la fonction $f_1$ et à partir de $f_1((0,...,0)) = 1$, comme explicité à la définition 5.9.

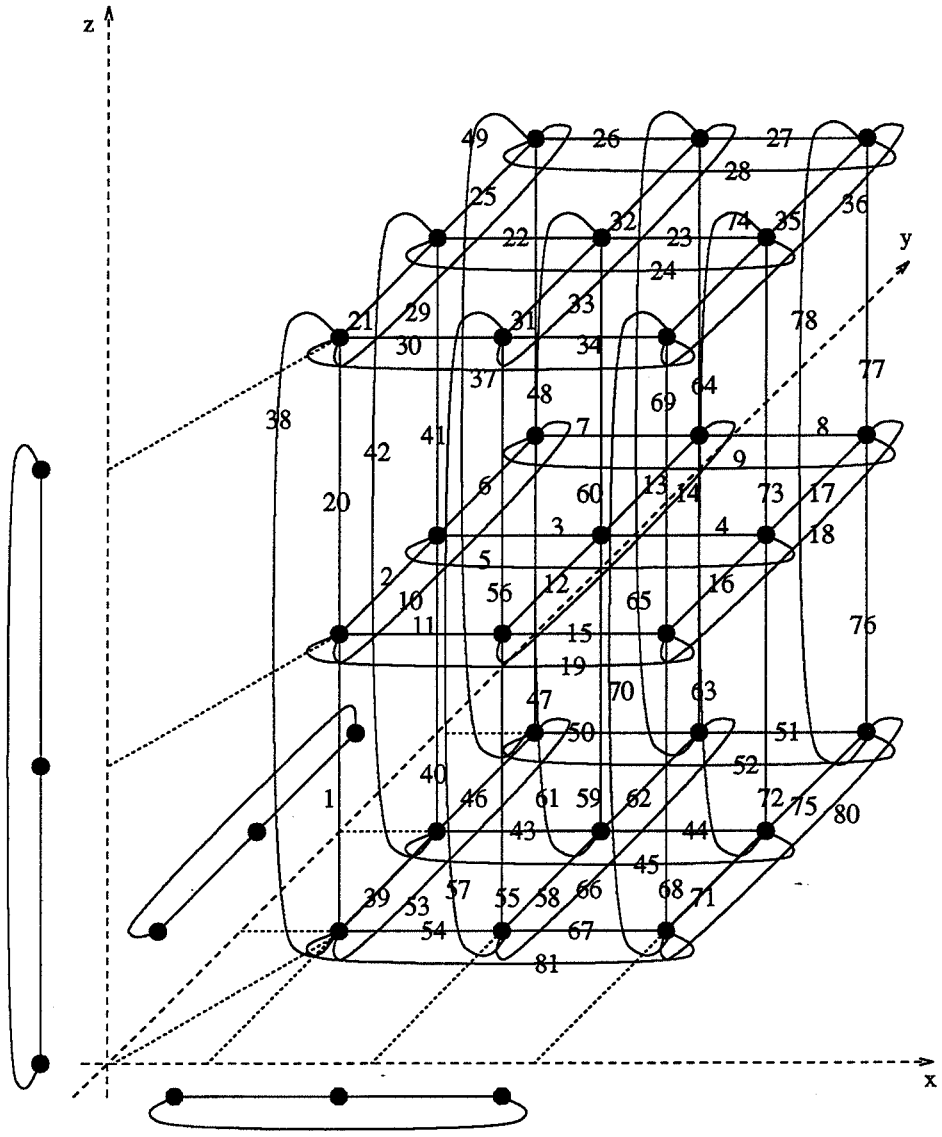


FIG. 5.8 – *Le parcours eulérien dans un tore(3,3)*

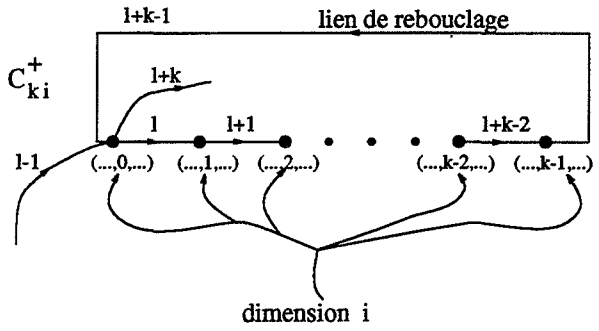


FIG. 5.9 – *Numérotation suivant la fonction $f_1$ d'un $C^+_{ki}$*

Du fait que le tore(d,k) est le produit cartésien de $d$ $k$-cycles, pour chaque dimension $i$, $i = 0...(d-1)$, le sens de parcours direct oriente tous les $k$-cycles de cette dimension $i$. Considérons les notations suivantes :

- on note $C_k$, un $k$-cycle,
- on note $C_{ki}$, un $k$-cycle de la dimension $i$,
- on note $C^+_{ki}$ (resp. $C^-_{ki}$), un $k$-cycle de la dimension $i$ orienté suivant le sens direct (resp. indirect).

La figure 5.9 montre le parcours d'un $C^+_{ki}$. Cette figure permet de voir toutes les dépendances permises dans un $C_{ki}$. En particulier, le routage sur ce $C_{ki}$ suivant un ordre croissant (resp. décroissant) pour le sens direct (resp. indirect) est permis. Notons également que la dépendance entre le lien de rebouclage (numéroté $l + k - 1$) et le premier lien (numéroté $l$), au niveau du nœud de coordonnée $(., x_{i-1} = 0, .)$ est interdite dans les deux sens, en dépit des règles de routage par cycle eulérien.

Par ailleurs, il est évident que la longueur d'un chemin minimal entre un nœud $s$ de coordonnées $(s_0, ..., s_{d-1})$ et un nœud $d$ de coordonnées $(d_0, ..., d_{d-1})$, est donnée par l'expression :

$$\sum_{i=0}^{d-1} Min(|C^+_{ki}(s_i, d_i)|, |C^-_{ki}(s_i, d_i)|)$$

où, $|C^+_{ki}(s_i, d_i)|$ (resp. $|C^-_{ki}(s_i, d_i)|$) est la longueur du chemin sur $C_{ki}$ de $s_i$ à $d_i$ suivant le sens direct (resp. indirect).

Nous faisons une preuve par récurrence sur le nombre $d$ de dimension du tore$(d,k)$ et nous montrons qu'au plus $(d-1)$ transitions de niveau sont nécessaires pour un routage optimal.

- $d=2$: Soit à router un message $m$ d'un nœud source $s=(s_0,s_1)$ à un nœud destination $d=(d_0,d_1)$. Nous distinguons plusieurs cas suivant les coordonnées des nœuds $s$ et $d$.
  1. $s_0=d_0$ ou $s_1=d_1$: le routage de $s$ à $d$ se fait sur un *seul* $C_{ki}$ $(i=0/1)$. Que l'on route suivant $C^+_{ki}$ ou $C^-_{ki}$, on aura au plus besoin d'une seule transition de niveau (au niveau du nœud tel que $x_{0/1}=0$),
  2. $s_0>d_0$ et $s_1\neq d_1$:
     (a) le plus court chemin sur la dimension 1 est suivant un $C^+_{k1}$:
        i. le plus court chemin sur la dimension 0 est suivant un $C^+_{k0}$: on route sur la dimension 0 suivant $C^+_{k0}$ de $s$ au nœud de coordonnées $(0,s_1)$ puis de ce nœud au nœud de coordonnées $(0,d_1)$ suivant $C^+_{k1}$ en montant (éventuellement) d'un niveau puis finalement de ce nœud vers le nœud $d$ suivant $C^+_{k0}$.
        ii. le plus court chemin sur la dimension 0 est suivant un $C^-_{k0}$: on route sur la dimension 0 suivant $C^-_{k0}$ de $s$ au nœud de coordonnées $(d_0,s_1)$ puis de ce nœud vers $d$ sur la dimension 1 suivant $C^+_{k1}$ en montant (éventuellement) de niveau.
     (b) le plus court chemin sur la dimension 1 est suivant un $C^-_{k1}$:
        i. le plus court chemin sur la dimension 0 est suivant un $C^+_{k0}$:
           A. $s_1>d_1$: on route sur la dimension 1 suivant $C^-_{k1}$ de $s$ au nœud de coordonnées $(s_0,d_1)$ puis de ce nœud vers $d$ sur la dimension 0 suivant $C^+_{k0}$ en montant (éventuellement) de niveau.
           B. $s_1<d_1$: on route sur la dimension 1 suivant $C^-_{k1}$ de s au nœud de coordonnées $(s_0,0)$, puis de ce nœud vers le nœud de coordonnées $(0,0)$ sur la dimension 0 suivant $C^+_{k0}$, puis de ce nœud sur la dimension 1 suivant $C^-_{k1}$ au nœud de coordonnées $(0,d_1)$ en montant de niveau (au niveau du nœud $(0,0)$), puis de ce dernier nœud vers $d$ sur la dimension 0 suivant $C^+_{k0}$.
        ii. le plus court chemin sur la dimension 0 est suivant un $C^-_{k0}$: on route sur la dimension 1 suivant $C^-_{k1}$ de $s$ au nœud de coordonnées $(s_0,d_1)$ en montant (éventuellement) de niveau puis de ce nœud vers $d$ sur la dimension 0 suivant $C^-_{k0}$.

3. $s_0 < d_0$ et $s_1 \neq d_1$ :
   (a) le plus court chemin sur la dimension 1 est suivant un $C^+_{k1}$ :
      i. le plus court chemin sur la dimension 0 est suivant un $C^+_{k0}$ : on route sur la dimension 0 suivant $C^+_{k0}$ de $s$ au noeud de coordonnées $(d_0, s_1)$, puis de ce nœud vers $d$ sur la dimension 1 suivant $C^+_{k1}$ en montant (éventuellement) de niveau.
      ii. le plus court chemin sur la dimension 0 est suivant un $C^-_{k0}$ :
         A. $s_1 > d_1$ : on route de $s$ sur la dimension 0 suivant un $C^-_{k0}$ au nœud de coordonnées $(0, s_1)$, puis de ce nœud sur la dimension 1 suivant un $C^+_{k1}$ au nœud de coordonnées $(0, 0)$, puis de ce nœud sur la dimension 0 suivant $C^-_{k0}$ au nœud de coordonnées $(d_0, 0)$ en montant de niveau (au niveau du nœud $(0, 0)$), puis de ce nœud vers d.
         B. $s_1 < d_1$ : on route de $s$ sur la dimension 0 suivant un $C^-_{k0}$ au nœud de coordonnées $(d_0, s_1)$ en montant (éventuellement) de niveau, puis de ce nœud sur la dimension 1 suivant $C^+_{k1}$ vers $d$.
   (b) le plus court chemin sur la dimension 1 est suivant un $C^-_{k1}$ :
      i. le plus court chemin sur la dimension 0 est suivant un $C^+_{k0}$ : on route de $s$ sur la dimension 1 suivant un $C^-_{k1}$ au nœud de coordonnées $(s_0, d_1)$ en montant (éventuellement) de niveau, puis de ce nœud sur la dimension 0 suivant $C^+_{k0}$ vers $d$.
      ii. le plus court chemin sur la dimension 0 est suivant un $C^-_{k0}$ :
         A. $s_1 > d_1$ : on route de $s$ sur la dimension 1 suivant un $C^-_{k1}$ au nœud de coordonnées $(s_0, d_1)$, puis de ce nœud sur la dimension 0 suivant $C^-_{k0}$ vers $d$ en montant (éventuellement) de niveau.
         B. $s_1 < d_1$ : on route de $s$ sur la dimension 0 suivant un $C^-_{k0}$ au nœud de coordonnées $(0, s_1)$, puis de ce nœud sur la dimension 1 suivant $C^-_{k1}$ vers le nœud de coordonnées $(0, d_1)$ en montant (éventuellement) de niveau puis de ce noeud vers $d$ sur la dimension 0 suivant $C^-_{k0}$.

- Supposons que la propriété est vraie jusqu'à l'ordre (d-1) et montrons qu'elle l'est également à l'ordre $d$. Soit à router un message $m$ d'un nœud source $s = (s_0, ..., s_{d-1})$ au nœud destination $d = (d_0, ..., d_{d-1})$, sur un chemin minimal, du tore$(d, k)$. Nous distinguons deux cas :
   1. le plus court chemin sur la dimension $(d-1)$ est suivant un $C^+_{k(d-1)}$ : Soit $j = (d_0, ..., d_{d-2}, s_{d-1})$ un nœud intermédiaire de routage. Les

nœuds $j$ et $s$ appartiennent au même tore$((d-1),k)$, défini par la coordonnée $x_{d-1} = s_{d-1}$.
Il est évident que le routage sur un plus court chemin de $s$ à $d$ dans un tore$(d,k)$, correspond à un routage sur un plus court chemin de $s$ à $j$ dans le tore$((d-1),k)$ défini par la coordonnée $x_{d-1} = s_{d-1}$ et à un routage sur un plus court chemin de $j$ à $d$ dans un $C_{k(d-1)}$.
D'après l'hypothèse de récurrence, le routage sur un plus court chemin de $s$ à $j$ dans le tore$((d-1),k)$ se fait en utilisant $(d-1)$ niveaux de communication. Nous pouvons donc faire au plus une et une seule autre transition de niveau pour le routage dans le tore$(d,k)$.
On route premièrement de $s$ à $j$ dans le tore$((d-1),k)$ en utilisant au maximum (d-1) niveaux, puis de $j$ à $d$ suivant $C^+_{k(d-1)}$.
Le problème se pose lorsqu'au niveau du nœud $j$, on doit monter de niveau et que le routage suivant $C^+_{k(d-1)}$ traverse le lien de rebouclage.
Pour monter de niveau au niveau de $j$, le dernier lien parcouru $l$ (juste avant le nœud $j$) doit être un lien direct (sinon, si c'est un lien indirect, la dépendance est permise et on ne monte pas de niveau). Cependant, d'après le parcours de cycle eulérien sur le tore$(d,k)$, $f_1(l)$ est inférieur au numéro suivant la fonction $f_1$, du premier lien parcouru dans $C^+_{k(d-1)}$, donc la dépendance est permise entre ces deux liens et on ne monte pas de niveau, au niveau de $j$. Ce qui veut dire que nous aurons au plus besoin d'un seul autre niveau de communication.

2. le plus court chemin sur la dimension $(d-1)$ est suivant un $C^-_{k(d-1)}$ :
Soit $j = (s_0, ..., s_{d-2}, d_{d-1})$ un nœud intermédiaire de routage. Les nœuds $j$ et $d$ appartiennent au même tore$((d-1),k)$, défini par la coordonnée $x_{d-1} = d_{d-1}$.
Il est évident que le routage sur un plus court chemin de $s$ à $d$ dans un tore$(d,k)$, correspond à un routage sur un plus court chemin de $s$ à $j$ dans un $C_{k(d-1)}$ et à un routage sur un plus court chemin de $j$ à $d$ dans le tore$((d-1),k)$ défini par la coordonnée $x_{d-1} = d_{d-1}$.
D'après l'hypothèse de récurrence, le routage sur un plus court chemin de $j$ à $d$ dans le tore$((d-1),k)$ se fait en utilisant $(d-1)$ niveaux de communication. Nous pouvons donc faire au plus une et une seule autre transition de niveau pour le routage dans le tore$(d,k)$.
On route premièrement de $s$ à $j$ suivant $C^-_{k(d-1)}$ en montant éventuellement de niveau lors du parcours de ce $C_{k(d-1)}$, puis de $j$ à $d$

dans le tore$((d-1),k)$ en utilisant au maximum (d-1) niveaux.
Le problème se pose lorsqu'au niveau du nœud $j$, on doit monter de niveau. Pour monter de niveau au niveau de $j$, le premier lien parcouru $l$ (juste après $j$) doit être un lien indirect (sinon, si c'est un lien direct, la dépendance est permise et on ne monte pas de niveau).
Cependant, d'après le parcours de cycle eulérien sur le tore$(d,k)$, $f_1(l)$ est inférieur au numéro suivant la fonction $f_1$ du lien parcouru juste avant $j$, donc la dépendance est permise entre ces deux liens indirects et on ne monte pas de niveau au niveau de $j$.

Finalement, dans tous les cas nous avons besoin de $d$ niveaux de communication pour router sur un chemin minimal de $s$ à $d$ dans le tore$(d,k)$. □

Ce théorème indique que, avec $d$ niveaux de communication, il existe *au moins un* chemin de communication optimal entre toute paire de nœuds *(source,destination)* dans un *tore*$(d,k)$. Le nombre de chemins optimaux est en effet certainement plus grand que 1.

L'algorithme de routage est représenté, au niveau de chaque nœud, par une table qui donne, pour chaque canal d'entrée et chaque destination, le ou les canaux de sortie qui réalisent le plus court chemin par rapport à la destination. Cette table est de taille $O(k^d)$. Le calcul de cette table est obtenu à l'aide d'un algorithme parallèle, inductif sur la longueur des chemins de communication. Nous détaillerons au chapitre suivant le calcul de ces tables de routage.

#### 5.5.2.2 Un second algorithme de routage dans les grilles toriques

Dans [50], au travers d'un autre parcours eulérien du *tore*$(d,k)$, nous montrons que 2 niveaux de communication sont suffisants pour un routage sur des chemins optimaux; en l'occurrence, un nombre de niveaux indépendant de la dimension du tore.

Nous proposons ensuite une amélioration de la méthode de routage par cycle eulérien, permettant de réduire le nombre de règles de routage de trois à une seule règle.

En utilisant cette amélioration et à partir du résultat précédent, nous en déduisons un algorithme de routage dans les grilles toriques multi-dimensionnelles, sans interblocage et tel que son graphe de dépendance est acircuitique, minimal, nécessitant un espace mémoire optimal pour la représentation de l'ensemble de l'information de routage et deux niveaux de communication pour le stockage temporaire des messages utilisateurs. Cependant, cet algorithme ne fournit qu'un

seul chemin de communication entre toute paire de nœuds (*source*, *destination*); il n'est donc pas adaptatif.

## 5.6 Application 2: La communication multi-niveaux selon le routage *e-cube*

Dans [52], nous présentons une autre application de la communication multi-niveaux aux grilles toriques multi-dimensionnelles.

Cette application est basée sur un schéma de communication primaire de type *e-cube*. Le routage d'un message $m$ depuis un nœud source de coordonnées $(s_0, s_1, ..., s_{d-1})$ à un nœud destination de coordonnées $(d_0, d_1, ..., d_{d-1})$ est effectué suivant un ordre décroissant de parcours des dimensions que le message doit traverser. Une transition de niveau a lieu, chaque fois que le message $m$ doit traverser un lien de rebouclage.

Le résultat est un algorithme de routage sans interblocage dont le graphe de dépendance est acircuitique, minimal, qui utilise un espace mémoire constant pour la représentation de l'information de routage et $d+1$ niveaux de communication pour une grille torique de dimension $d$. Cependant cet algorithme ne permet qu'un seul chemin de communication entre toute paire de nœuds, il n'est donc pas adaptatif.

## 5.7 Interprétation des résultats

Les applications qui viennent d'être décrites pour la grille torique, valident bien l'efficacité de la communication multi-niveaux, vis-à-vis de l'objectif fixé initialement.

En effet, au travers de deux schémas de communication primaires particuliers, nous avons conçu trois algorithmes de routage pour ce réseau. Ces algorithmes satisfont beaucoup des critères de routage énoncés dans le chapitre 2 de la première partie.

Au vu de la bibliographie étudiée, parmi les stratégies de routage générales; i.e. qui ne sont pas dédiées à des topologies particulières de réseaux d'interconnexion, nous ne connaissons pas de stratégie permettant d'obtenir des propriétés de routage pour la grille torique, similaires à l'algorithme décrit dans [50]*. Principalement, un routage optimal, sans interblocage et tel que le graphe de dépendance induit est acircuitique, et qui utilise un espace mémoire constant pour les deux composantes mémoire du routage.

L'intêret pour la grille torique est justifié par diverses raisons. En effet, parmi

les réseaux d'interconnexion statiques, réguliers, usuels des architectures massivement parallèles, donc principalement l'hypercube, la grille et le tore, ce dernier est l'unique réseau qui n'admet pas de routage "ad hoc".

Le terme "ad hoc" étant dans le sens d'un routage avec un chemin unique, sans interblocage, optimal, *"plat"*; i.e. qui nécessite un niveau de communication, et dont la fonction de routage est représentée par une expression simple. Le routage *e-cube* vérifie toutes ces propriétés pour l'hypercube. Le routage $XY$ les vérifie pour la grille. Pour la grille torique, l'algorithme de routage décrit dans [20], ne peut pas être considéré comme tel à cause des deux raisons suivantes:

1. Son graphe de dépendance contient des cycles,

2. L'expression représentant la fontion de routage n'est pas simple. Elle requiert un temps polynomial en $k$.

L'algorithme que nous avons décrit dans [50]*, vérifie l'ensemble de ces propriétés et utilise une expression simple pour le calcul des chemins de communication. Notons toutefois que même en utilisant 2 niveaux de communication, cet algorithme est *"plat"*. En effet, un routage optimal sans interblocage pour la grille torique nécessite au minimum deux niveaux de communication [20].

Aussi, la communication multi-niveaux est bien représentative de la caractérisation d'une stratégie de routage, que nous avons donnée dans la première partie. En effet, nous avons distingué les deux composantes que sont:

1. la fonction de routage; c'est le schéma *SCP* qui spécifie pour tout couple de nœud un chemin de communication,

2. un mécanisme de virtualisation. ce sont les niveaux de communication qui permettent l'intégration des critères d'efficacité.

Elle représente également une bonne technique de virtualisation. En effet, comparée, par exemple, à la méthode de "Comptage de sauts"[8] [43], c'est une approche inverse qui est prise.

En ce sens, dans cette dernière stratégie de routage, le mécanisme de virtualisation est utilisé pour assurer l'absence d'interblocage et non pas un routage sur des chemins optimaux. Le graphe de tampons acircuitique assure que le routage est sans interblocage. La fonction de routage consiste ensuite en une fermeture transitive du réseau. Alors que pour la communication multi-niveaux, la technique de virtualisation est utilisée pour l'obtention des chemins optimaux.

La méthode de "Comptage de sauts" utilise systématiquement le diamètre du réseau tampons par canal de communication, en particulier pour le tore. C'est à

8. Qui permet également un facteur d'élongation unitaire.

dire, $n\lfloor k/2 \rfloor$ tampons par canal. Ce qui est prohibitif par rapport à la communication multi-niveaux basée sur le routage par cycle eulérien.

## 5.8 Conclusion

Dans ce chapitre, nous avons présenté une méthode générale pour l'amélioration des critères de routage d'un schéma de communication donné. Cette méthode est assez générique pour envisager une corrélation entre d'autres critères de correction et d'efficacité. En effet, l'idée de la généricité est selon l'optique suivante : un niveau de communication permet d'assurer les critères de correction, donc les propriétés de complétude et d'absence d'interblocage, la hiérarchisation de ces niveaux permet ensuite d'intégrer un ou plusieurs critères d'efficacité.

C'est par exemple une approche bien adaptée au problème de la tolérance aux pannes physiques tant que, bien entendu, le graphe du réseau d'interconnexion reste connexe. En ce sens, supposons qu'un message $m$ arrivé par un canal d'entrée $c_e$ doive emprunter un canal de sortie $c_s$ vers sa destination $d$, supposons que le lien de communication du canal $c_s$ soit en panne, il suffit alors d'envoyer le message $m$ sur n'importe quel canal de sortie robuste, en transitant d'un niveau de communication, si la dépendance $(c_e, c_s)$ est interdite, et en restant dans le même niveau sinon.

Cette méthode soulève cependant le problème de *l'attraction du dernier niveau de communication.* En effet, les messages tendent à monter vers le dernier niveau. Ils induisent ainsi une surcharge des tampons de ce niveau, qui *"travaillent plus"*, par rapport aux tampons des autres niveaux.

Par ailleurs, étant donné un réseau de communication, si le nombre de niveaux de communication ne permet pas un facteur d'élongation unitaire pour un schéma *SCP* donné, toutes les élongations de chemin vont se produire au dernier niveau de communication. Une question qui se pose alors dans ce cas, vis-à-vis de la qualité du facteur d'élongation est : *est-il préférable de gérer les transitions de niveau suivant une autre politique que celle de conserver un routage sur un chemin optimal, ou est-il préférable d'avoir toutes les élongations de chemin au dernier niveau de communication?.*

# Chapitre 6

# Implantation de la communication multi-niveaux

## 6.1 Une implantation distribuée

La communication multi-niveaux basée sur le routage par cycle eulérien, a été développée en simulation sur un réseau de TRANSPUTERS T805 d'une machine VOLVOX, pilotée par une station SUN. Nous avons utilisé le langage C PARALLEL INMOS [1].

Cette implantation a été réalisée de façon *totalement distribuée*, directement sur le réseau de processeurs cibles. Les tables de coût et de routage de chaque processeur sont calculées *localement* par le processeur lui-même. Ce calcul est conduit suivant un processus itératif permettant de converger vers un état final, représentant les tables adéquates de chaque processeur. Aucun chargement de ces tables n'est effectué à partir d'un processeur *hôte* [90] ; ainsi nous obtenons un système *auto-constructif*.

Pour calculer les tables de coût et de routage, chaque processeur exécute le même programme. Le programme commence par reconnaître l'identificateur du processeur sur lequel il est exécuté.

Un processeur est simulé par un processus et chaque processus admet son propre identificateur. Le programme reconnaît en fait l'identificateur du processus auquel il est associé. Nous plongeons le réseau de communication simulé, dont la topologie peut être quelconque et dont les nœuds sont des processus, dans le réseau de TRANSPUTERS. Chaque TRANSPUTER gère de ce fait plusieurs processus. Nous avons utilisé plusieurs TRANSPUTERS[1] du réseau physique pour pouvoir simuler des réseaux d'interconnexion de taille assez grande. Les processus

1. Au maximum 5.

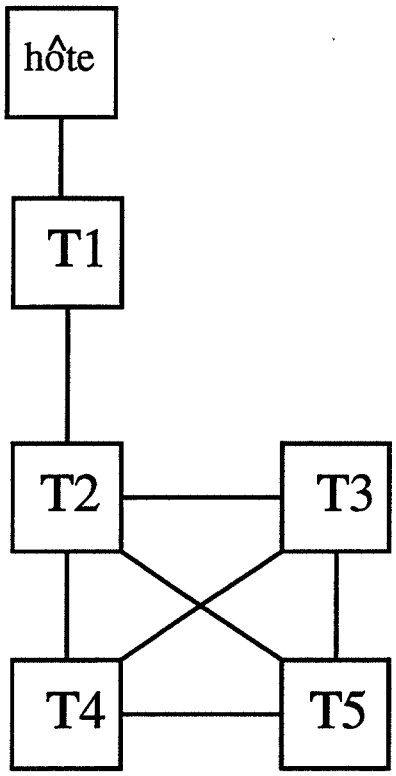


FIG. 6.1 – *Le réseau de* TRANSPUTERS T805

communiquent via des canaux de communication logiques [1].

Dans l'état actuel, le programme utilise des canaux de communication logiques, avec un maximum de 5 TRANSPUTERS du réseau physique, comme sur la figure 6.1. Par rapport à cette figure, les processus simulant les nœuds du réseau de communication, sont placés sur les TRANSPUTERS T2, T3, T4 et T5. Comme on le voit sur cette figure, ces quatre TRANSPUTERS sont complètement reliés entre eux. Les canaux logiques sont placés par le système sur les liens physiques, et quelle que soit alors la paire de processus du réseau simulé, ces processus peuvent communiquer.

Après avoir reconnu l'identificateur du processeur[2] qui lui associé, le programme identifie ensuite l'ensemble de ses processeurs voisins. Il détermine pour chaque canal de sortie[3], l'identificateur du processeur auquel il est relié par ce canal. L'ensemble des voisins d'un processeur $s$ sont alors stockés dans un vecteur

2. Dans la suite, nous parlerons de processeur au lieu de processus.
3. Logique.

local $V_s$.

Du fait que le schéma de communication primaire utilise un cycle eulérien du réseau d'interconnexion des processeurs, et du fait également que le système est implanté de manière totalement distribuée, nous avons développé un algorithme distribué de calcul d'un tel cycle dans un réseau de processeurs [10]. Cet algorithme a été intégré et implanté sur le réseau de TRANSPUTER T805.

## 6.2 Un algorithme distribué de calcul d'un cycle eulérien dans un réseau

L'algorithme que nous décrivons dans ce paragraphe est bien adapté au principe suivant lequel l'implantation a été développée. Au niveau d'un nœud du réseau, cet algorithme ne suppose pas une *connaissance globale* du réseau. Au contraire, il suppose uniquement la connaissance de l'ensemble des canaux de communication du nœud en question.

### 6.2.1 Le principe de l'algorithme

Soit $G = (S, A)$ le graphe qui modélise le réseau d'interconnexion des processeurs. On suppose que tous les nœuds de $G$ sont de degré pair. On rappelle qu'un parcours eulérien de $G$, est un parcours fermé, initié à partir d'un nœud racine $r$, qui passe par toutes les arêtes une et une seule fois.

Le principe de l'algorithme consiste à décrire un tel parcours de manière distribuée. Pour cela, nous utilisons une *numérotation* locale des canaux de communication de chaque sommet $s$, avec les entiers allant de 1 à $deg_s$ où $deg_s$ est le degré du sommet $s$. Cette numérotation génère un parcours eulérien de $G$ dans le sens suivant : *au niveau d'un sommet $s$, le successeur dans le parcours eulérien de $G$, d'un canal d'entrée numéroté $l$ est le canal de sortie numéroté $l + 1$.*

L'algorithme est basé sur la propriété suivante de la théorie des graphes [8]:

**Propriété 6.1 :** *Tout cycle $\mu$ d'un graphe $G$ est somme de cycles élémentaires.*

On peut étendre cette propriété et remarquer qu'un cycle d'un graphe est somme de cycles non forcément élémentaires. En particulier, un parcours eulérien d'un graphe $G$ peut être vu comme une somme de cycles.

En effet, l'algorithme construit un ensemble de cycles à chaque *étape*, puis les joint entre eux, pour obtenir tout le parcours eulérien à la fin de l'exécution de celui-ci.

Dans ce but, l'algorithme utilise deux types de messages, SATURER et VERIFIER qui vont traverser tous les canaux de communication de $G$ et générer à la

fin de l'exécution de l'algorithme, la numérotation locale adéquate des canaux de communication de chaque nœud.

L'algorithme est basé sur un *raisonnement récursif.* Il est initié par un nœud quelconque $r_1$, appelé la *première racine.* Tous les autres nœuds sont dans l'état de *l'attente de la réception* d'un message.

Initialement, tous les canaux de communication de chaque nœud sont numérotés zéro. Chaque nœud $s$, maintient un compteur local $C_s$ initialisé à zéro et utilisé pour le numérotage des canaux. A la réception (resp. l'émission) d'un message SATURER sur un canal d'entrée (resp. de sortie) $c$, ce compteur est incrémenté et sa valeur courante est affectée au canal $c$ comme numéro.

Pour des raisons de clarté d'exposé, dans ce qui suit, lorsqu'on dit qu'un nœud $s$ envoie (resp. reçoit) un message SATURER sur le canal de sortie (resp. d'entrée) $c$, cela veut dire également que le canal $c$ est affecté d'un numéro.

La première racine $r_1$ choisit (aléatoirement) un canal de sortie et envoie sur ce canal le message SATURER. Par une transmission de nœuds adjacents en nœuds adjacents, ce message va parcourir un *certain circuit* du graphe $G$, en provoquant la numérotation adéquate de l'ensemble des canaux traversés, et revient à la première racine $r_1$ [4]. Nous appelons un tel parcours une *première étape de* $r_1$.

Lorsque le message SATURER revient à la racine $r_1$, on distingue deux cas :

1. Il reste des canaux de sortie non numérotés de $r_1$ : ce dernier choisit alors un autre canal (non numéroté) et envoie le message SATURER sur ce canal de sortie. Comme à l'étape précédente, ce message va parcourir un autre circuit du réseau et revenir au nœud $r_1$, en ayant causé la numérotation adéquate de tous les canaux traversés. Ceci est une deuxième étape à partir de $r_1$, et ainsi de suite...,

2. Il ne reste plus de canaux de sortie non numérotés au niveau de $r_1$ : on dira alors que le nœud $r_1$ est *saturé.* Dans ce cas, le nœud $r_1$ envoie un message VERIFIER sur le canal de sortie numéroté pendant sa première étape. Ceci dans le but de *vérifier que les nœuds appartenant au circuit construit pendant cette étape sont saturés.* Le nœud qui va recevoir ce message, va alors devenir la *racine courante* (i.e. la deuxième racine $r_2$) et effectuera le même traitement qui a été précédemment effectué par la racine $r_1$. Il va *saturer* ses canaux de communication et va envoyer ensuite le message VERIFIER sur son canal de sortie numéroté pendant sa première étape. Ceci illustre le raisonnement récursif de l'algorithme.

Lorsque le message VERIFIER revient au nœud $r_1$ par le canal d'entrée numéroté pendant la première étape, le nœud $r_1$ va l'envoyer sur le canal de sortie

4. Puisque le graphe $G$ est connexe.

numéroté pendant la deuxième étape, et ainsi de suite... . L'algorithme termine lorsque la première racine $r_1$ reçoit le message VERIFIER sur son canal d'entrée, dont le numéro correspond à son degré.

Une description des différents cas possibles qu'un nœud peut rencontrer est comme suit.

- Pour le nœud $r_1$ (la première racine) :
  - le nœud $r_1$ envoie le message SATURER sur l'un de ses canaux de communication,
  - si le nœud $r_1$ reçoit un message SATURER sur un de ses canaux, il l'envoie sur un de ses canaux non déjà numérotés s'il lui en reste. Sinon, il envoie un message VERIFIER sur le canal numéroté 1.
  - si le nœud $r_1$ reçoit un message VERIFIER sur un canal de numéro $j$, il l'envoie sur le canal de numéro $j+1$, si $j < deg_{r_1}$. Si $j = deg_{r_1}$ l'algorithme termine[5].
- Pour un nœud $s$ (autre que la première racine) :
  - si un nœud $s$ reçoit un message SATURER sur un de ses canaux de communication, il l'envoie sur un de ses canaux non déjà numérotés s'il lui en reste. Si tous ses canaux sont numérotés, le nœud $s$ envoie un message VERIFIER sur le canal qui a le plus petit numéro et qui n'a pas encore été traversé par un tel message,
  - lorsqu'un nœud $s$ reçoit un message VERIFIER sur son canal numéroté 1, deux cas se présentent :
    i) *tous ses canaux de communication sont numérotés* . Dans ce cas, le nœud $s$ envoie le message VERIFIER sur son canal numéroté 2. De plus, si le degré de $s$ est 2, il termine.
    ii) *le nœud $s$ a encore des canaux de communication qui ne sont pas encore numérotés*. Soit $S$ le nombre de canaux de $s$ qui sont déjà numérotés (leur numéro varie donc de 1 à $S$). Le nœud $s$ :
      1) modifie la numérotation des canaux de communication qui sont déjà numérotés de la façon suivante : le canal numéroté $S$ sera numéroté $deg_s$[6], le canal numéroté $S-1$ sera numéroté $deg_s - 1$,..., le canal numéroté 2 sera numéroté $deg_s - S + 2$,

5. Notons que seule la première racine $r_1$ peut recevoir le message VERIFIER sur le canal ayant le numéro $deg_{r_1}$.

6. Le degré de $s$.

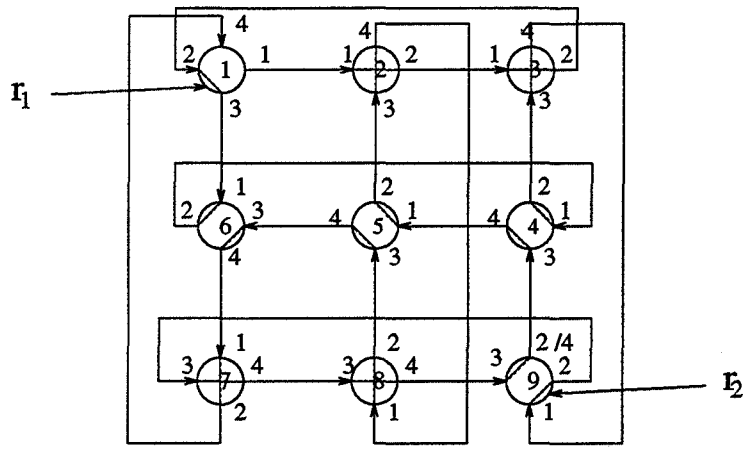


FIG. 6.2 – *Une exécution de l'algorithme sur un tore(2,3)*

2) choisit aléatoirement un de ses canaux qui n'est pas encore numéroté, soit $y$,

3) envoie un message SATURER sur le canal $y$. Ce canal aura pour étiquette 2.

– Lorsqu'un nœud $s$ reçoit un message VERIFIER sur son canal de numéro $\ell$ ($\ell \neq 1$), il l'envoie sur son canal de numéro $\ell + 1$. De plus, si $\ell + 1 = deg_s$ et que tous les canaux de $s$ ont été visités par le message VERIFIER, alors l'algorithme au niveau du nœud $s$ termine.

Sur la figure 6.2, nous avons représenté une exécution de l'algorithme sur une grille torique de dimension 2 et de base 3. La première racine $r_1$ qui initie la construction du parcours eulérien est le nœud de numéro 1.

Pour les besoins de la preuve de l'algorithme donné au paragraphe suivant, nous faisons la remarque suivante :

**Remarque 6.1.** Supposons que tous les canaux de communication de la deuxième racine $r_2$ ont été numérotés pendant les étapes de la première racine $r_1$. Ce qui est équivalent à dire que la deuxième racine $r_2$ a été saturée pendant les étapes de la première racine $r_1$. Dans ce cas, lorsque le nœud $r_1$ envoie le message VERIFIER sur le canal de sortie numéroté pendant sa première étape, le nœud $r_2$ va directement envoyer ce message sur son canal de sortie numéroté durant la première étape de $r_1$. Le même traitement est effectué à chaque réception du message VERIFIER par le nœud $r_2$. Ainsi, le

nœud $r_2$ ne va jamais générer un message de type SATURER après la réception d'un message de type VERIFIER. C'est-à-dire qu'il ne va jamais initier la construction d'un circuit du graphe $G$.

Alors que s'il existe des canaux de sortie de $r_2$ qui n'ont pas été numérotés durant les étapes de $r_1$, le nœud $r_2$ va effectivement générer un message SATURER lorsqu'il va recevoir le message VERIFIER pour la première fois. En l'occurrence, il va réellement initier la construction d'un certain circuit du graphe $G$.

Ainsi, pour les besoins de la preuve de l'algorithme, un nœud sera référé comme un nœud racine, s'il initie effectivement la construction d'un certain circuit du graphe $G$. En ce sens, un nœud racine est défini comme un nœud qui envoie un message SATURER juste après la réception d'un message VERIFIER.

### 6.2.2 Correction de l'algorithme

Pour prouver la correction de l'algorithme, nous devons montrer que:

1. D'une part, tous les canaux de communication de chaque nœud sont numérotés et que la numérotation obtenue correspond bien à un parcours eulérien du réseau, dans le sens que nous avons précédemment défini,

2. D'autre part, au niveau de chaque nœud du réseau, l'algorithme termine.

**Théorème 6.1 :** *Soit $G$ le graphe connexe associé à un réseau d'interconnexion $I$. Si le graphe $G$ est eulérien, l'algorithme décrit ci-dessus construit un parcours eulérien de $G$ et termine.*

**Preuve :** Nous faisons une preuve par récurrence sur le nombre, $n_r$, de nœuds racines, générés par une exécution de l'algorithme sur le graphe $G$.

1. $\underline{n_r = 1}$ : Dans ce cas, il existe un seul nœud racine; la première racine $r_1$, qui initie la construction du parcours eulérien. Ceci implique que pendant l'exécution de l'algorithme sur $G$, il n'existe pas de nœud qui reçoit un message VERIFIER et envoie un message SATURER. Par conséquent, le message VERIFIER est généré pour la première et dernière fois par la première racine $r_1$.

   Soient $C_1, C_2, ..., C_k$, les circuits du graphe $G$ traversés pendant les étapes $1, ..., k$ du nœud $r_1$. Pendant ces étapes, tous les canaux de communication appartenant à ces circuits sont correctement numérotés.

Lorsque le message SATURER retournera au nœud $r_1$ à l'étape $k$, ce dernier enverra le message VERIFIER sur le canal de sortie de l'étape 1. Ce message va traverser les circuits $C_1, C_2, ..., C_k$ exactement dans le même ordre que le message SATURER et retournera au nœud $r_1$ qui terminera.

Tout nœud appartenant aux circuits $C_1, C_2, ..., C_k$ a été saturé, puisqu'il a reçu le message VERIFIER et n'a pas généré un message SATURER (d'après l'hypothèse $n_r = 1$). Par ailleurs, tous les canaux d'un tel nœud ont été correctement numérotés puisque, chaque fois que le nœud reçoit le message SATURER sur le canal d'entrée d'un circuit $C_i$, il l'envoie sur le canal de sortie du même circuit, en provoquant la numérotation adéquate des deux canaux.

De plus, tout nœud appartenant aux circuits $C_1, C_2, ..., C_k$ termine, puisque le message VERIFIER suit exactement le même parcours que le message SATURER et lorsqu'un tel nœud envoie le message VERIFIER sur le canal de sortie dont le numéro correspond à son degré, ce nœud termine.

Il reste à montrer que tout sommet du graphe $G$ appartient à au moins un des circuits $C_1, C_2, ..., C_k$. C'est-à-dire que tout sommet reçoit au moins une fois le message SATURER.

Supposons le contraire; i.e. supposons qu'il existe un sommet $i$ de $G$ qui n'a jamais reçu le message SATURER. Considérons les sommets $v_1, v_2, ..., v_m$, voisins de $i$. Il n'est pas difficile de voir que si le sommet $i$ n'a jamais reçu le message SATURER, alors *aucun* de ses sommets voisins n'est *saturé*. Par conséquent, aucun de ces sommets n'appartient à l'un des circuits $C_1, C_2, ..., C_k$; puisque tout sommet appartenant à l'un de ces circuits est saturé. En l'occurrence, chacun des sommets $v_1, v_2, ..., v_m$ ne reçoit jamais le message SATURER.

Considérons alors les sommets voinsins de chaque sommet $v_j$, pour $j = 1, ..., m$. Comme pour le sommet $i$, l'unique façon pour qu'un sommet $v_j$ ne reçoive jamais le message SATURER, est que ses propres sommets voisins ne le reçoivent jamais. Ainsi, en répétant ce raisonnement de sommets voisins en sommets voinsins, et du fait que le graphe $G$ est connexe, on en déduit qu'il existe au moins un sommet voisin de la première racine $r_1$ qui ne reçoit jamais le message SATURER. Ceci est une contradiction, puisque le sommet $r_1$ a été saturé et que donc tous ses sommets voisins ont reçu le message SATURER.

Finalement, tout sommet de $G$ appartient à au moins un des circuits $C_1, C_2, ..., C_k$, et l'algorithme construit bien un parcours eulérien et

termine.

2. $\underline{n_r = n \Longrightarrow n_r = n+1}$ : Supposons que le théorème est vrai lorsqu'une exécution de l'algorithme sur un graphe génère $n$ nœuds racines. Soit $G$ un graphe tel qu'une exécution de l'algorithme génère $n+1$ nœuds racines. Soient $r_1, r_2, ..., r_{n+1}$ ces nœuds racines, générés dans cet ordre[7]. Considérons le nœud $r_{n+1}$. D'après la remarque 6.1, ce nœud a reçu un message VERIFIER, à un point donné d'exécution de l'algorithme, et a envoyé un message SATURER sur un de ses canaux non encore numérotés. Ceci implique que ce nœud a auparavant reçu au moins une fois le message SATURER, pendant une étape de l'un des $n$ premiers nœuds racines, et l'a envoyé sur un canal non numéroté. Ainsi, le nœud $r_{n+1}$ appartient à au moins un circuit généré par l'un des $n$ premiers nœuds racines. Soit $C$ ce circuit.

   Lorsque le nœud $r_{n+1}$ reçoit le message VERIFIER pour la première fois, il initie la construction d'un circuit en envoyant un message SATURER sur un de ses canaux non numérotés. Ceci représente sa première étape. Au même moment, il renumérote ses canaux déjà numérotés suivant la description ci-dessus. En effet, il envoie le message SATURER sur son canal de sortie de l'étape 1 qui va être numéroté 2. Ce message va parcourir un circuit du graphe $G$ et revenir au nœud $r_{n+1}$ sur un canal d'entrée qui va être numéroté 3. S'il reste encore des canaux non numérotés, le nœud $r_{n+1}$ va alors effectuer la même procédure, et ainsi de suite... . Soient $C_1, C_2, ..., C_k$, les circuits parcourus pendant les étapes du nœud $r_{n+1}$.

   Lorsque le message SATURER est reçu sur le canal d'entrée du circuit $C_k$, à la dernière étape $k$, le nœud $r_{n+1}$ envoie alors le message VERIFIER sur le canal de sortie du circuit $C_1$. Ce message va alors parcourir les circuits $C_1, C_2, ..C_{k-1}$, et $C_k$ dans cet ordre et retournera au nœud $r_{n+1}$. Ce dernier l'envoie ensuite sur le canal de sortie du circuit $C$ et termine.

   Pendant les étapes 1 à $k$ de la racine $r_{n+1}$, tous les canaux appartenant aux circuits $C_1, C_2, ..C_k$ sont correctement numérotés. Chaque fois qu'un nœud appartenant à l'un de ces circuits reçoit le message SATURER sur un canal d'entrée, il l'envoie sur un canal de sortie, en effectuant la numérotation adéquate des deux canaux. Le même traitement est effectué pour le message VERIFIER. Chaque nœud appartenant à l'un de ces circuits, ne fait que recevoir ce message sur un canal d'entrée et l'envoyer sur un canal de sortie. Si un tel nœud n'appar-

7. $r_{n+1}$ est la dernière racine.

tient qu'aux circuits $C_1$ et/ou $C_2$,..., et/ou $C_k$, ce nœud va terminer après avoir envoyé le message VERIFIER sur le canal de sortie de plus grand numéro. Sinon, si un tel nœud appartient à au moins un circuit de l'une des $n$ premières racines $r_1, r_2, ..., r_n$, le nœud ne termine pas.

Considérons à présent la procédure suivante : *tous les liens de communication appartenant aux circuits $C_1, C_2, ..., C_k$ sont enlevés du graphe G. Soit G', le sous-graphe restant. On considère alors sur le graphe G' la même exécution - que sur le graphe G - de l'algorithme précédent. Cette exécution va engendrer n nœuds racines; les nœuds $r_1, r_2, ..., r_n$. D'après l'hypothèse de récurrence, cette exécution de l'algorithme construit un parcours eulérien de G' et termine. On en déduit alors que cette exécution sur G, construit un parcours eulérien et termine.*

Ceci achève la preuve du théorème 6.1 □

### 6.2.3 Le code de l'algorithme

Nous donnons dans ce paragraphe le code exécuté par chaque nœud $i$, dans un langage proche du PASCAL. On suppose que les liens de communication du nœud $i$ sont identifiés par des entiers appartenant à l'ensemble $\{1, ..., deg_i\}$.

La procédure **Construire_Cycle_Eulerien** est le programme principal exécuté par le nœud qui initie la construction du cycle eulérien. La procédure **Cycle_Eulerien** est le programme principal exécuté par n'importe quel autre nœud.

Au niveau de chaque nœud, sont également implantées deux procédures :

- La procédure **Recevoir_SATURER(c)** : elle correspond au traitement effectué lors de la réception d'un message de type SATURER sur le canal $c$,
- La procédure **Recevoir_VERIFIER(c)** : elle correspond au traitement effectué lors de la réception d'un message de type VERIFIER sur le canal $c$.

**const**

$deg_i = \ldots$; {* initialisé au degré du nœud $i$ *}

**var**

$label_i, Num\_channel_i$ : $array[1..deg_i]$ of integer;
$cpt_i$, $cpv_i$, $c$, $j$, $x$, $y$ : integer;
$constructing_i$ : boolean;

**function Canal_Libre();**

{* lorsque cette procédure est appellée par un nœud $i$, elle retourne un canal non numéroté, choisi de manière aléatoire, si un tel canal existe. S'il n'existe pas, elle retourne -1 *}

**Program Construire_Cycle_Eulerien;**

{* Le programme principal de la racine qui initie la construction du cycle eulérien *}

```
begin
   cpt_r := 0; cpv_r := 0; constructing_r := true;
   for j:=1 to deg_r do label_r[j] := 0;
   for j:=1 to deg_r do Num_channel_r[j] := 0;
   x := Canal_Libre(); cpt_r := cpt_r + 1; label_r[x] := cpt_r;
   Num_channel_r[cpt_r] := x;

   send(SATURER) over the channel x;
   while constructing do
      begin
        receive message over a channel c;
        case message of
             SATURER : Recevoir_SATURER(c);
             VERIFIER : Recevoir_VERIFIER(c);
        endcase
      end
end
```

**Program Cycle_Eulerien;**

{* Le programme principal d'un sommet i ($i \neq r$) *}

begin

$cpt_i := 0$; $cpv_i := 0$; $constructing_i := true$;

for $j$:=1 to $deg_i$ do $label_i[j] := 0$;

for $j$:=1 to $deg_i$ do $Num\_channel_i[j] := 0$;

while *constructing do*

begin

receive *message over a channel c;*

case *message* of

*SATURER : Recevoir_SATURER(c);*

*VERIFIER : Recevoir_VERIFIER(c);*

endcase

end

end

**Procedure Recevoir_SATURER(k);**

{* Procédure correspondant à la réception d'un message *SATURER* sur le cannal $k$ *}

begin

$cpt_i := cpt_i + 1$; $label_i[k] := cpt_i$; $Num\_channel_i[cpt_i] := k$;

$x := Canal\_Libre()$;

if $x \neq -1$ then begin

$cpt_i := cpt_i + 1$; $label_i[x] := cpt_i$; $Num\_channel_i[cpt_i] := x$;

*send(SATURER)* over the channel $x$;

end

else begin

$cpv_i := cpv_i + 1$; $y := Num\_channel_i[cpv_i]$;

*send(VERIFIER)* over the channel $y$;

end

endif

end

**Procedure Recevoir_VERIFIER($\ell$);**

{* Procédure correspondant à la réception d'un message *VERIFIER* sur le canal $\ell$ *}

```
begin
  cpv_i := cpv_i + 1;
  if cpv_i = deg_i then constructing_i := false
                   else begin x := Canal_Libre();
                              if x ≠ -1
                                then begin
                                       for j := cpt_i downto 2 do
                                           begin
                                           c := Num_channel_i[j]; label_i[c] := deg_i + j - cpt_i;
                                           end
                                       cpt_i := 2; label_i[x] := cpt_i; Num_channel_i[cpt_i] := x;
                                       send(SATURER) over the channel x;
                                       end
                                else begin
                                       cpv_i := cpv_i + 1; y := Num_channel_i[label_i[ℓ] + 1];
                                       send(VERIFIER) over the channel y;
                                       if cpv_i = deg_i then constructing_i := false
                                       end
                              endif
                        end
  endif
end
```

### 6.2.4 Les performances de l'algorithme

D'après la description ci-dessus de l'algorithme, chaque arête du graphe $G$ est traversée par deux messages exactement : un message SATURER et un message VERIFIER. Aussi, lorsqu'une arête est traversée dans un sens par un message de type SATURER, elle est également traversée par un message de type VERIFIER dans le même sens. Elle n'est traversée par aucun message dans l'autre sens.

Le nombre total de messages utilisés par l'algorithme est donc de $2|A|$ où, $|A|$ est le cardinal de l'ensemble $A$. Ceci correspond également à la complexité en temps de l'algorithme. En effet, le parcours eulérien est calculé en $2|A|$ "*grands pas*" d'exécution.

Cet algorithme ne nécessite pas l'échange d'un grand nombre de messages. Sa complexité en message est relativement bonne. Cependant, sa complexité en temps peut être améliorée. En effet, on peut facilement remarquer, qu'à un instant donné, un seul nœud du réseau est dans un état actif, i.e. il effectue un certain traitement. Les autres nœuds sont dans l'état d'attente de la réception d'un message.

Ainsi, une direction d'amélioration de cet algorithme consiste en une "*parallélisation*" de celui-ci, afin de permettre une meilleure complexité en temps.

## 6.3 Le calcul des tables de coût et de routage

Après avoir calculé la structure de routage du schéma primaire, chaque processeur $s$ détermine pour chaque canal d'entrée[8] $c_e$, l'ensemble des canaux de sortie $c_s$ tels que l'envoi, suivant $c_s$, d'un message arrivé à $s$ par $c_e$ est permis par la méthode de routage par cycle eulérien. Le programme calcule en fait, pour chaque canal d'entrée $c_e$, la table $V_s^E(c_e)$, représentant l'ensemble des sommets $v$, voisins de $s$, tels que la dépendance entre le canal $c_e$ et le canal $(s, v)$ est permise par la méthode de routage par cycle eulérien.

Au niveau d'un sommet $s$, la fonction de routage est définie par :

$$\begin{aligned} F_s : A \times N \times S &\longrightarrow \wp(A \times N) \\ (c_e, n_c, d) &\longmapsto (c_s, n_s) \end{aligned}$$

Cependant, au niveau implantation, cette fonction a été traduite sous forme de deux tables, à deux entrées chacune :

1. $LM_s(c_e, d)$ : cette table donne, pour le canal d'entrée $c_e$ et la destination $d$, l'ensemble de tous les canaux de sortie $c_s$ correspondant au routage selon la communication multi-niveaux basée sur le cycle eulérien.

8. Nous rappelons qu'un canal est une voie de communication unidirectionnelle.

2. $NM_s(c_e, d)$ : cette table donne, pour un routage par la communication multi-niveaux basée sur le cycle eulérien, le nombre de transitions de niveau sur le chemin de $s$ à $d$ d'un message arrivé à $s$ par $c_e$.

Nous avons intégré également, le calcul de la table de routage $LE_s(c_e, d)$ :

3. $LE_s(c_e, d)$ : cette table donne, pour le canal d'entrée $c_e$ et la destination $d$, l'ensemble de tous les canaux de sortie $c_s$ correspondants au routage par la méthode du cycle eulérien.

Le calcul de l'ensemble de ces tables est obtenu à l'aide d'une technique de *programmation dynamique* [89, 72]. En effet, pour chaque stratégie de routage, pour permettre les plus courts chemins de communication possibles, nous avons utilisé une méthode d'optimisation dynamique dont le critère d'optimisation porte sur le coût d'une communication, en terme de distance, d'un message arrivé à un sommet $s$ par un canal d'entrée $c_e$ et à une destination $d$. L'ensemble des coûts calculés sont :

1. $CE_s(c_e, d)$ : représente la distance d'un chemin de communication minimal de $s$ à $d$ pour le canal d'entrée $c_e$, lorsqu'on route par la méthode du cycle eulérien.

2. $CM_s(c_e, d)$ : représente la distance d'un chemin de communication minimal de $s$ à $d$ pour le canal d'entrée $c_e$, lorsqu'on route par la communication multi-niveaux basée sur le routage par cycle eulérien.

3. $C_s(c_e, d)$ : représente la distance d'un chemin de communication minimal de $s$ à $d$ pour le canal d'entrée $c_e$, lorsqu'on route suivant la fermeture transitive du réseau. En l'occurrence, la distance minimale de $s$ à $d$ dans le graphe $G$.

Pour chacun de ces coûts, la technique d'optimisation utilise une *induction sur la longueur des chemins de communication*, pour obtenir au dernier *pas d'induction*, les coûts minimaux. En effet, si l'on note :

- $C_s^{(i-1)}(c_e, d)$ : le coût de la communication de $s$ à $d$ en $i-1$ pas d'induction, pour un routage suivant la fermeture transitive du réseau,

il est évident [2, 89] que le coût en $i$ pas d'induction est donné par l'équation suivante :

$$C_s^{(i)}(c_e, d) = Min(C_s^{(i-1)}(c_e, d), 1 + \underset{v \in V_s}{Min}(C_v^{(i-1)}((s, v), d))) \qquad (I)$$

De même, lorsque la structure de routage est le cycle eulérien, nous avons que :

$$CE_s^{(i)}(c_e,d) = Min(CE_s^{(i-1)}(c_e,d), 1 + \underset{v \in V_s^E(c_e)}{Min}(CE_v^{(i-1)}((s,v),d))) \qquad (II)$$

et le calcul de la table $LE_s^{(i)}(c_e,d)$ à l'étape $i$, se déduit ensuite par l'équation suivante :

$$LE_s^{(i)}(c_e,d) = \begin{cases} LE_s^{(i-1)}(c_e,d) & si\ \ CE_s^{(i-1)}(c_e,d) < CE_s^* \\ LE_s^{(i-1)}(c_e,d) \bigcup \underset{v \in V_s^{E*}(c_e)}{(\cup\{(s,v)\})} & si\ \ CE_s^{(i-1)}(c_e,d) = CE_s^* \\ \underset{v \in V_s^{E*}(c_e)}{\cup\{(s,v)\}} & sinon \end{cases} \qquad (III)$$

où,

$$CE_s^* = 1 + \underset{v \in V_s^E(c_e)}{Min}(CE_v^{(i-1)}((s,v),d))$$

et,

$$V_s^{E*}(c_e) = \{v \in V_s^E(c_e)\ permettant\ CE_s^*\}$$

Le même raisonnement est utilisé pour le schéma de communication général. En effet, si $n_n$ est le nombre de niveaux de communication, le coût de la communication, à l'étape $i$, d'un message arrivé à $s$ par le canal $c_e$ et à destination $d$ est donné par l'équation :

$$CM_s^{(i)}(c_e,d) = Min(CM_s^{(i-1)}(c_e,d), 1 + \underset{v \in V_s^*(c_e,d)}{Min}(CM_v^{(i-1)}((s,v),d))) \qquad (IV)$$

où,

$$V_s^*(c_e,d) = \{v \in V_s\ /NM_s^{(i-1)}(c_e,v) + NM_v^{(i-1)}((s,v),d) \leq (n_n - 1)\}$$

i.e. le coût minimum entre le coût à l'étape $(i-1)$ et le coût minimum, pris sur l'ensemble des voisins $v$ de $s$ tel que la nombre total de transitions de niveau de $s$ à $v$ et de $v$ à $d$ soit inférieur à $(n_n - 1)$, à l'étape $(i-1)$ d'un tel voisin $v$. L'équation décrivant les transitions de niveau s'écrit :

$$NM_s^{(i)}(c_e,d) = \{NM_s^{(i-1)}(c_e,v) + NM_v^{(i-1)}((s,v),d)\}\ si\ \exists\ v \in V_s^*(c_e,d)\ tel\ que\ CM_s^{(i)} \qquad (V)$$

A partir de ces deux équations, on déduit l'équation de mise à jour de la table de routage $LM_s$ à l'étape $i$ à partir de celle de l'étape $(i-1)$,

$$LM_s^{(i)}(c_e,d) = \begin{cases} LM_s^{(i-1)}(c_e,d) & si \;\; CM_s^{(i-1)}(c_e,d) < CM_s^* \\ LM_s^{(i-1)}(c_e,d) \bigcup (\bigcup_{v \in V_s^*(c_e,d)} \{(s,v)\}) & si \;\; CM_s^{(i-1)}(c_e,d) = CM_s^* \\ \bigcup_{v \in V_s^*(c_e,d)} \{(s,v)\} & sinon \end{cases} \qquad (VI)$$

où,

$$CM_s^* = 1 + \underset{v \in V_s^*(c_e,d)}{Min}(CM_v^{(i-1)}((s,v),d))$$

Les équations qui viennent d'être écrites, indiquent simplement comment passer du pas d'induction $(i-1)$ au pas $i$. Cette induction part de l'étape 0, représentant en fait les initialisations adéquates des tables de coût et de routage. Nous décrivons, les valeurs initiales de toutes les tables en $OCCAM$ :

*BEGIN*

*/* initialisations pour la fermeture transitive */*

*SEQ*

*FOR* $(d \in S)$

*FOR* $(v \in V_s)$

$$C_s^{(0)}((v,s),d) = \begin{cases} 0 & si \;\; s = d \\ 1 & si \; d \in V_s - \{v\} \\ \infty & sinon \end{cases}$$

*/* initialisations pour le routage par cycle eulérien */*

*FOR* $(d \in S)$

*FOR* $(v \in V_s)$

$$CE_s^{(0)}((v,s),d) = \begin{cases} 0 & si \;\; s = d \\ 1 & si \; d \in V_s^E((v,s)) \\ \infty & sinon \end{cases}$$

;

$$LE_s^{(0)}((v,s),d) = \begin{cases} \emptyset & si \;\; s = d \\ \{(s,d)\} & si \;\; d \in V_s^E((v,s)) \\ \emptyset & sinon \end{cases}$$

*/* initialisations pour la communication multi-niveau */*

*FOR* $(d \in S)$

*FOR* $(v \in V_s)$

$$CM_s^{(0)}((v,s),d) = \begin{cases} 0 & si \;\; s = d \\ 1 & si \; d \in V_s - \{v\} \\ \infty & sinon \end{cases}$$

;

$$NM_s^{(0)}((v,s),d) = \begin{cases} 0 & si \;\; s = d \\ 0 & si \; d \in V_s^E((v,s)) \\ 1 & si \; d \in V_s - V_s^E((v,s)) - \{v\} \\ 0 & sinon \end{cases}$$

;

$$LM_s^{(0)}((v,s),d) = \begin{cases} \emptyset & si \;\; s = d \\ \{(s,d)\} & si \; d \in V_s - \{v\} \\ \emptyset & sinon \end{cases}$$

*END*

L'ensemble des itérations $(I)$, $(II)$, $(IV)$ et $(V)$ convergent en au plus le diamètre du réseau pas d'induction [72]. Les tables de coût et de routage se stabilisent

alors[9]. Notons $d(G)$ ce diamètre. Par ailleurs, soit *Calculer_nouveaux_couts*(), la procédure permettant le calcul suivant l'équation $(I)$ des coûts à l'étape $i$. Le programme de calcul des coûts pour la fermeture transitive du réseau s'écrit alors en *OCCAM* comme suit :

```
BEGIN                    /* phase de calcul */
  SEQ
    k:= 1;
    WHILE (k ≤ d(G)) DO
      SEQ
        PAR (v ∈ Vs)
          PAR (v ∈ Vs)
            (s,v) ! Cs(.,.)
          PAR (v ∈ Vs)
            (v,s) ? Cs(.,.)
        Calculer_nouveaux_couts();
        k:= k+1;
END
```

De même, soit *Calculer_nouveaux_couts_Euler*() (resp. *Calculer_nouveaux_canaux_Euler*()), la procédure permettant le calcul suivant l'équation $(II)$ (resp. $(III)$) des coûts (resp. des canaux de sortie), lorsque la structure de routage est le cycle eulérien. Le programme de calcul des tables de coût et de routage pour la méthode du cycle eulérien est :

```
BEGIN                    /* phase de calcul */
  SEQ
    k:= 1;
    WHILE (k ≤ d(G)) DO
      SEQ
        PAR (v ∈ Vs)
          PAR (v ∈ Vs)
            (s,v) ! CEs(.,.)
          PAR (v ∈ Vs)
            (v,s) ? CEs(.,.)
        Calculer_nouveaux_couts_Euler();
        Calculer_nouveaux_canaux_Euler();
        k:= k+1;
END
```

9. Pour notre implantation distribuée, nous ne calculons pas le diamètre du réseau. Nous utilisons en fait le nombre de nœuds du réseau.

Finalement, soit $Calculer\_nouveaux\_couts - niveaux\_Multi - niveaux()$ la procédure permettant le calcul suivant les équations $(IV)$ et $(V)$ des coûts et du nombre de transitions de niveau, pour le routage suivant la communication multi-niveaux basée sur le cycle eulérien. Soit $Calculer\_nouveaux\_canaux\_Multi - niveaux()$ la procédure de calcul des canaux de sortie suivant l'équation $(VI)$. Nous avons alors le programme suivant pour la communication multi-niveaux :

```
BEGIN                    /* phase de calcul */
  SEQ
    k:= 1;
    WHILE (k ≤ d(G)) DO
      SEQ
        PAR (v ∈ Vs)
          PAR (v ∈ Vs)
            (s,v) ! CMs(.,.)
          PAR (v ∈ Vs)
            (v,s) ? CMs(.,.)
          PAR (v ∈ Vs)
            (s,v) ! NMs(.,.)
          PAR (v ∈ Vs)
            (v,s) ? NMs(.,.)
        Calculer_nouveaux_couts - niveaux_Multi - niveaux();
        Calculer_nouveaux_canaux_Multi - niveaux();
        k:= k+1;
END
```

A ce niveau de l'exposé, il est important de noter que l'implantation de la communication multi-niveaux que nous venons de décrire, tient compte et même résout le problème de l'attraction des niveaux de communication supérieurs, que nous avons relevé dans la conclusion du chapitre précédent.

En effet, ce problème survient pour une implantation *"directe"* qui *"calque"* directement le principe de la communication multi-niveaux sur le réseau de communication, tel qu'il a été décrit dans le chapitre précédent. Cependant, pour notre implantation, ceci n'est pas le cas.

Pour un nombre $n_n$ donné, de niveaux de communication, l'équation $(IV)$ qui régit le calcul des tables de coûts de communication[10], et par là des tables de routage, permet de *contrôler* les transitions de niveaux en fonction justement du nombre $n_n$.

10. Pour la communication mutli-niveaux basée sur le routage par cycle eulérien.

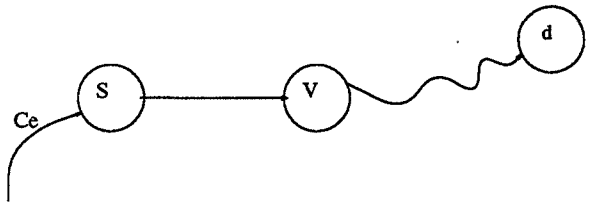


FIG. 6.3 – *Interprétation de l'équation (IV)*

L'equation $(IV)$ est :

$$CM_s^{(i)}(c_e, d) = Min(CM_s^{(i-1)}(c_e, d), 1 + Min_{v \in V_s^*(c_e,d)}(CM_v^{(i-1)}((s, v), d)))$$

où,

$$V_s^*(c_e, d) = \{v \in V_s \ / NM_s^{(i-1)}(c_e, v) + NM_v^{(i-1)}((s, v), d) \leq (n_n - 1)\}$$

$$NM_s^{(i)}(c_e, d) = \{NM_s^{(i-1)}(c_e, v) + NM_v^{(i-1)}((s, v), d)\} \ si \ \exists \ v \in V_s^*(c_e, d) \ tel \ que \ CM_s^{(i)} \quad (V)$$

Sur la figure 6.3, nous avons représenté une interprétation de cette équation. Au niveau du sommet $s$, pour le calcul du coût de la communication à l'étape $i$, du routage d'un message arrivé par le canal $c_e$ et à destination $d$, on *"prend"* le coût minimum entre,

1. le coût à l'étape $(i-1)$ et,
2. 1 plus le coût minimum, à l'étape $(i-1)$, calculé sur l'ensemble des voisins $v$ tel que, *le nombre de transitionx de niveau, à l'étape* $(i-1)$, *de* $s$ *à* $v$ *et de* $v$ *à* $d$ *reste inférieur à* $(n_n - 1)$.

Ainsi, même si un voisin $v'$ de $s$ réalise, à l'étape $i$, un coût inférieur au coût que réalise un voisin $v$, mais induit un nombre de transitions de niveau supérieur à $(n_n - 1)$, ce voisin n'est pas comptabilisé.

Le programme que nous avons implanté calcule également, pour chaque stratégie de routage, les longueurs des chemins de communication minimaux entre toute paire de processeurs $(s, d)$, indépendamment du canal d'entrée. Les tables calculées sont :

$$C_s(d) = Min_{v \in V_s}(C_s((v, s), d))$$

$$CE_s(d) = \underset{v \in V_s}{Min}(CE_s((v,s),d))$$

$$CM_s(d) = \underset{v \in V_s}{Min}(CM_s((v,s),d))$$

Finalement, le programme calcule les facteurs d'élongation suivants :

$$\underset{(s,d)\in S^2}{\text{Max}}\left(\frac{CM_s(d)}{C_s(d)}\right) \quad , \quad \underset{(s,d)\in S^2}{\text{Max}}\left(\frac{CE_s(d)}{C_s(d)}\right) \quad , \quad \underset{(s,d)\in S^2}{\text{Max}}\left(\frac{CE_s(d)}{CM_s(d)}\right)$$

# Chapitre 7

# Un schéma de communication primaire

## 7.1 Introduction

Dans le chapitre 5, nous avons décrit une méthode générale pour l'amélioration des critères de routage d'un schéma de communication donné. Nous avons constaté que cette méthode est basée sur un schéma $SCP$, qui avait la priorité de résoudre *efficacement* le problème de l'interblocage. En plus de la satisfaction de la propriété de complétude, ce schéma primaire doit utiliser tous les liens de communication et doit utiliser également un tampon pour le stockage temporaire des messages par canal de communication.

Dans ce chapitre, nous décrivons un schéma *SCP* général qui cible, spécifiquement, le problème de l'interblocage. En plus des critères énumérés ci-dessus, qu'il satisfait, nous verrons que ce schéma intègre davantage de critères d'efficacité.

Bien que ce schéma s'applique d'une manière générale, indépendamment de la technique de commutation, il est particulièrement adapté à la technique du "wormhole". En raison du fait que, pour cette technique de commutation, il existe une relation bijective entre les canaux de communication et les tampons de stockage des messages.

Pour prévenir l'interblocage, ce schéma de communication est basé sur le théorème de DALLY et SEITZ, que nous rappelons ici :

**Théorème [23, 24]** : *Une fonction de routage $F$ sur un réseau d'interconnexion $I$ est sans interblocage* **ssi** *son graphe de dépendance $D_F$ est acircuitique.*

Nous rappelons également que le graphe de dépendance $D_F$ est un graphe orienté dont les sommets sont les canaux de communication de $I$ et les arcs sont les couples $(c_e, c_s)$ tel que le canal d'entrée $c_e$ *dépende* du canal de sortie $c_s$. La

relation de dépendance entre un canal d'entrée $c_e$ et un canal de sortie $c_s$ est donnée par la définition suivante :

**Définition :** *Un canal d'entrée $c_e$ dépend d'un canal de sortie $c_s$ si $c_s$ est un successeur direct de $c_e$ dans un chemin de communication induit par $F$ sur $I$.*

Etant basé sur ce théorème, ce schéma de communication admet un graphe de dépendance acircuitique. Comme nous l'avons souligné dans la première partie de ce rapport, cette condition est nécessaire pour l'obtention d'une stratégie de routage robuste.

Dans la première partie, nous avons également constaté que la notion de *dépendance* entre deux canaux de communication successifs est à la base d'une méthode préventive de résolution de l'interblocage, qui nécessite un seul tampon par canal de communication. Plus une fonction de routage sur un réseau d'interconnexion $I$ permet de dépendances, plus elle peut intégrer potentiellement des critères d'efficacité.

Pour réaliser un tel objectif, une bonne approche préventive consisterait donc à considérer la *structure de routage* du réseau d'interconnexion $I$, qui contient **toutes** les dépendances entre canaux de communication, pour en extraire de cet ensemble, un sous ensemble qui n'induit pas de circuit d'interblocage.

Une telle approche constitue une orientation radicale par rapport à l'approche classique qui consiste, à partir d'une structure de routage *particulière* du réseau, i.e. un réseau couvrant, comme par exemple un arbre couvrant, et à router sur cette structure suivant des règles de routage sans interblocage. En effet, pour de telles approches, il existe des dépendances entre canaux de communication qui sont éliminées, même si certaines n'induisent pas d'interblocage.

En partant de la structure de routage du réseau de communication qui contient toutes les dépendances, la méthode que nous présentons dans ce chapitre essaye de maximiser le nombre de dépendances à permettre. Nous l'illustrons d'abord sur un réseau sous forme d'un anneau bidirectionnel.

## 7.2 Routage sans interblocage dans un anneau

La figure 7.1 illustre le résultat de la méthode sur un réseau sous forme d'un anneau de processeurs. Sur cette figure, la dépendance $(c_1, c_2)$ entre le canal d'entrée $c_1$ et le canal de sortie $c_2$, au niveau du processeur $p_1$, est éliminée. Cela, pour prévenir une situation d'interblocage correspondant à un parcours suivant le sens positif[1] du réseau. De même pour le sens négatif, la dépendance

1. Représenté sur la figure.

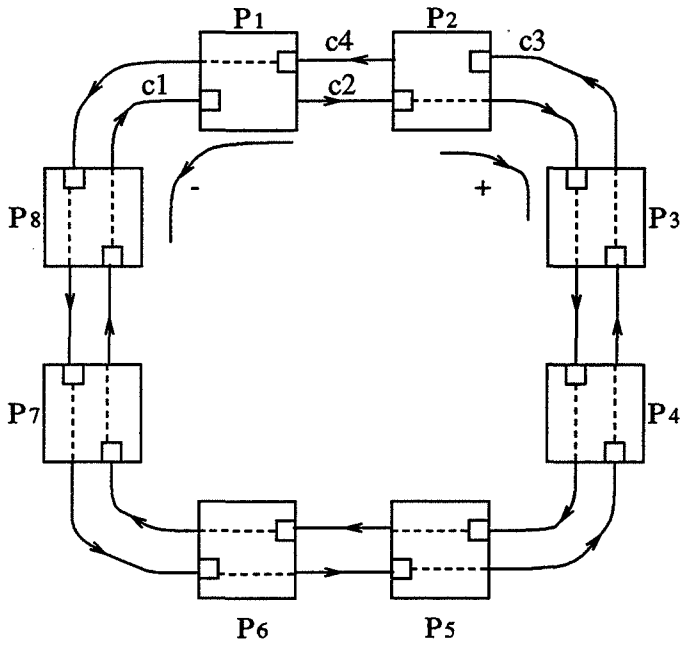


FIG. 7.1 – *Routage sans interblocage dans un anneau*

$(c_3, c_4)$ entre le canal d'entrée $c_3$ et le canal de sortie $c_4$ est éliminée, au niveau du processeur $p_2$, pour prévenir un interblocage dans ce sens.

La fonction de routage induite par l'ensemble des dépendances permises sur cet anneau satisfait à la propriété de complétude. En effet, quelle que soit la paire de processeur $(p_i, p_j)$, il existe un chemin de communication de $p_i$ à $p_j$ et de $p_j$ à $p_i$. Cependant ces chemins de communication ne sont pas forcément minimaux.

Ce qu'il est important de noter sur la figure 7.1, c'est que la prévention de l'interblocage n'est pas basée sur le mécanisme de virtualisation. En effet, aucune forme de réplication des éléments du réseau n'est nécessaire pour un routage sans interblocage. En particulier, un canal physique n'est pas partagé en canaux virtuels et la méthode n'induit qu'un seul tampon par canal physique.

Dans la section qui suit, nous montrons que dans le cas d'un réseau quelconque, un routage sans interblocage nécessitant un tampon par canal de communication est possible.

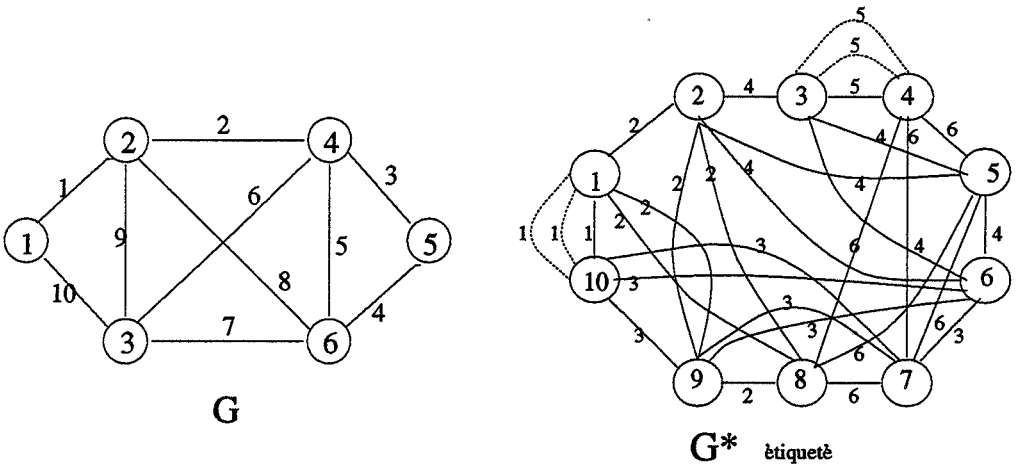


FIG. 7.2 – *Un graphe G et son graphe G**

## 7.3 La méthode de dérivation de graphe de dépendance acircuitique

### 7.3.1 Définitions et Notations

Soit $G = (S, A)$ le graphe qui modélise le réseau d'interconnexion des processeurs.

**Définition 7.1 :**

Considérons le graphe **représentatif des sommets** de $G$ [8], $G^* = (A, A')$ :

- L'ensemble $A$ des sommets de $G^*$ est constitué par les arêtes de $G$,
- Un couple de sommets $(s_1, s_2)$ de $G^*$ est relié par une arête, ssi le couple d'arêtes qu'ils représentent dans $G$, admettent un sommet d'adjacence commun.

Considérons la définition suivante :

**Définition 7.2 :**
Soient $f_1$ (resp. $f_2$) une *numérotation*[2] des sommets (resp. des arêtes) de $G$ :

$$f_1 : S \longrightarrow \{1, ..., |S|\}$$
$$s \longmapsto f_1(s)$$

$$f_2 : A \longrightarrow \{1, ..., |A|\}$$
$$a \longmapsto f_2(s)$$

La fonction $f_2$ induit directement un étiquetage des sommets du graphe $G^*$. En effet, un sommet de $G^*$ est étiqueté par le numéro donné par la fonction $f_2$ de l'arête de $G$ qu'il représente.

La fonction $f_1$ induit également un étiquetage des arêtes de $G^*$ suivant la politique suivante : une arête du graphe $G^*$ est étiquetée par le numéro du sommet d'adjacence dans $G$ qu'elle représente (voir fig 7.2).

Dans la suite de ce chapitre, les nœuds et les arêtes de $G$ (resp. $G^*$) seront identifiés par leurs étiquettes par rapport aux applications $f_1$ et $f_2$.

Dans la terminologie que nous utilisons, nous rappelons qu'une arête correspond à un concept non orienté et représente dans le graphe $G$ un lien de communication. Un arc correspond à un concept orienté et représente dans $G$ un canal de communication.

### 7.3.2 Le principe de la méthode

Le principe de la méthode consiste à *exploiter* l'information contenue dans le graphe représentatif des sommets, $G^*$ vis-à-vis du problème de l'interblocage. En effet, $G^*$ *modélise* **toutes** *les dépendances entre paires d'arcs* $(a_e, a_s)$[3] qu'une fonction de routage $F$ sur le réseau de graphe $G$, peut induire.

Considérons dans $G^*$ un couple de sommets $(s_1, s_2)$ reliés par une arête $a$. C'est à dire dans $G$, deux arêtes $s_1$ et $s_2$ adjacentes au niveau d'un sommet $a$. Cette adjacence implique la possibilité d'une dépendance de la forme :

- $(s_1, s_2)$, entre l'arc d'entrée $s_1$ et l'arc de sortie $s_2$, au niveau du sommet $a$,

  et/ou,

- $(s_2, s_1)$ entre l'arc d'entrée $s_2$ et l'arc de sortie $s_1$, au niveau du sommet $a$,

2. Quelconque.
3. *(arc d'entrée, arc de sortie).*

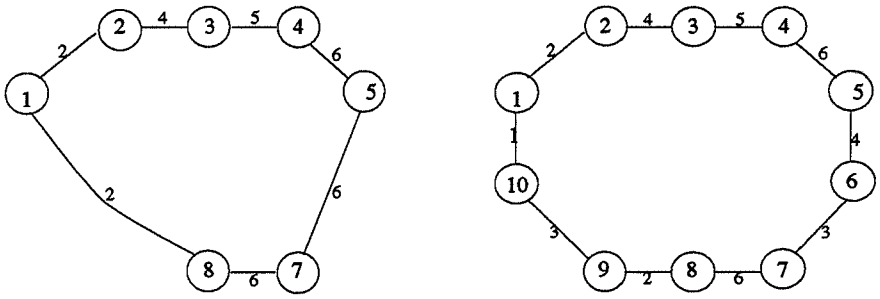


FIG. 7.3 – *Un parcours non blocant et un parcours d'interblocage*

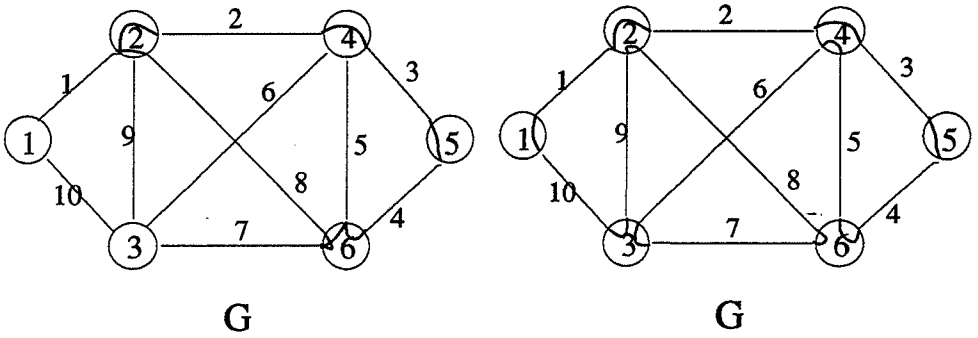


FIG. 7.4 – *Les suites de dépendances correspondantes dans G*

pouvant être induite par la fonction de routage $F$ sur $G$.

Plus précisément, dans $G^*$, l'orientation de l'arête $a$ du sommet $s_1$ vers le sommet $s_2$, i.e. l'arc $(s_1, s_2)$ représente dans $G$, la dépendance $(s_1, s_2)$ entre l'arc d'entrée $s_1$ et l'arc de sortie $s_2$ au niveau du sommet $a$, que la fonction $F$ peut induire.

De même, dans $G^*$, l'orientation de l'arête $a$ du sommet $s_2$ vers le sommet $s_1$, i.e. l'arc $(s_2, s_1)$ représente dans $G$, la dépendance $(s_2, s_1)$ entre l'arc d'entrée $s_2$ et l'arc de sortie $s_1$ au niveau du sommet $a$, que la fonction $F$ peut induire.

Ainsi, conceptuellement, un *circuit* de $G^*$ -c'est-à-dire un parcours fermé d'une suite d'arcs de $G^*$- correspond à un parcours fermé d'une suite de dépendances dans le graphe $G$. Ce parcours étant fermé, il correspond donc à un état d'interblocage potentiel dans $G$. Par un raisonnement similaire, un cycle de $G^*$ représente deux états d'interblocage potentiel dans $G$; un pour chaque sens de parcours de ce cycle.

La figure 7.3 de droite illustre un cycle du graphe $G^*$. Nous avons représenté sur la figure 7.4 de droite, la suite fermée de dépendances dans $G$ correspondantes à ce cycle. Sur cette figure, on voit bien l'état d'interblocage induit. Plus précisement, les deux états d'interblocage induits, l'un correspondant à un parcours dans le sens inverse de l'autre.

D'après le théorème de DALLY&SEITZ, pour *dériver* une fonction de routage $F$ sans interblocage sur $G$, il *suffit* alors d'éliminer dans $G^*$ un ensemble d'arcs[4], de sorte que le graphe résultant soit sans circuit. Ce graphe représentera alors le graphe de dépendance $D_F$ de la fonction $F$, qui est ensuite déduite.

Ainsi, le principe simple de la méthode consiste à calculer un graphe de dépendance $D_F$ sans circuit, à partir du graphe représentatif des sommets de $G$, et d'en déduire ensuite la fonction de routage sans interblocage $F$.

Cependant, une remarque concernant les cycles de $G^*$ s'impose. En effet, considérons le cycle de $G^*$ représenté sur la figure 7.3 de gauche. Ce cycle ne correspond pas à un parcours de cycles d'interblocage dans le graphe $G$; il ne définit pas deux états d'interblocage potentiel dans $G$ . Nous avons représenté sur la figure 7.4 de gauche les dépendances correspondantes à ce cycle. Comme on peut le remarquer, ces dépendances ne définissent pas deux interblocages potentiels dans $G$. En effet, sur la figure 7.3 de gauche, on peut constater trois arêtes consécutives qui ont pour étiquette 6 et deux arêtes consécutives qui ont pour étiquette 2.

Il faut donc déterminer la caractéristique des cycles du graphe $G^*$ qui définissent un interblocage potentiel dans le graphe $G$. La proposition suivante décrit cette caractéristique :

4. Donc, un ensemble de dépendances dans $G$.

**Proposition 7.1 :** *Un cycle du graphe $G^*$ définit une situation d'interblocage potentiel dans le graphe $G$ ssi deux arêtes successives de ce cycle n'ont jamais la même étiquette.*

**Preuve** ($\Leftarrow$) : Soit $\mu$ un cycle de $G^*$ qui satisfait à cette propriété. A partir d'un sommet $s$ quelconque, soit un parcours de ce cycle tel que à chaque arête (resp. sommet) on associe l'étiquette du sommet (resp. arête) qui vient juste d'être parcouru. C'est à dire que l'on effectue une rotation d'un "cran", dans le sens du parcours, entre les étiquettes de sommets et arêtes adjacents. Le nouvel étiquetage définit un cycle du graphe $G$. Ce cycle correspond à un interblocage potentiel dans $G$.

($\Rightarrow$) : Soit un état d'interblocage dans le graphe $G$. Nous avons donc une boucle de demande d'utilisation de tampons pleins entre un ensemble $s_i, i = 1...m$ de sommets de $G$. Considérons le cycle du graphe $G$ correspondant à cette boucle. Soit un parcours de ce cycle, tel que à chaque arête (resp. sommet) on associe l'étiquette du sommet (resp. arête) qui vient juste d'être parcouru. Le nouvel étiquetage définit un cycle de $G^*$ qui satisfait à la propriété énoncée dans la proposition. □

En effet, si deux arêtes consécutives d'un cycle de $G^*$ ont la même étiquette, ces deux arêtes modélisent l'adjacence entre trois arêtes de $G$, au niveau du *même sommet.* On peut voir ce cas, sur la figure 7.4 de gauche, au niveau du sommet 6.

Sur la base de la proposition 7.1, nous pouvons donner une formulation du problème de la dérivation d'une fonction de routage $F$ sans interblocage à partir du graphe représentatif des sommets, $G^*$ comme suit :

**Formulation 7.1 :** *Soustraire du graphe $G^*$ un ensemble d'arcs qui élimine* **tous** *les circuits correspondants à un interblocage potentiel dans le graphe $G$.*

C'est là une première formulation du problème posé. En effet, on peut essayer d'optimiser la solution et chercher à éliminer dans $G^*$ un ensemble *minimum* d'arcs, afin de permettre un ensemble *maximum* de dépendances pour la fonction de routage $F$ sans interblocage ainsi dérivée. Ce qui est l'objectif recherché pour le schéma $SCP$, à travers lequel l'intégration des critères d'efficacité est potentiellement possible. La nouvelle formulation du problème est alors :

**Formulation 7.2 :** *Soustraire du graphe $G^*$ un ensemble d'arcs qui n'élimine* **que et tous** *les circuits correspondant à un interblocage potentiel dans le graphe $G$.*

Le problème de la minimisation du nombre de dépendance à éliminer est répertorié *NP-Complet.* En effet, on peut montrer qu'il contient le problème de la recherche de chemins dans un graphe orienté muni de paires de sommets interdites, problème répertorié *NP-Complet* dans [35].

La formulation 7.2 représente ainsi une spécification *NP-Complete* du problème que nous voulons résoudre. Les solutions optimales ne sont donc pas envisageables. Nous proposons dans le paragraphe qui suit une heuristique pour le calcul du graphe de dépendance $D_F$.

### 7.3.3 Une heuristique pour le calcul du graphe de dépendance

Une autre manière de poser le problème consiste à prendre l'approche inverse et formuler le problème comme suit :

**Formulation 7.3 :** *A partir d'un graphe de dépendance $D_F$ vide, inclure un ensemble maximum d'arcs du graphe $G^*$ qui n'induisent pas d'interblocage potentiel dans le graphe $G$.*

Par rapport à la formulation 7.3, le point crucial est *l'ordre d'inclusion* des arcs de $G^*$ dans le graphe de dépendance $D_F$. En effet, le choix de l'ordre suivant lequel ces arcs sont inclus dans le graphe de dépendance influe directement sur le nombre de dépendances obtenues à la fin. Cette incidence est due au fait que, d'une part, chaque arc du graphe $G^*$ appartient à plusieurs circuits de ce graphe et d'autre part, au fait que le nombre de circuits n'est pas le même pour tous les arcs.

En ce qui concerne l'heuristique que nous décrivons dans ce paragraphe, cet ordre est induit par un parcours particulier du graphe $G^*$. En effet, supposons que le graphe $G$ est régulier et que tous ses sommets sont de *même degré* $\delta$. Les sommets de $G^*$ sont alors *tous* de degré $2(\delta - 1)$. Cette *parité* du degré des sommets de $G^*$ induit l'existence d'un *parcours eulérien* de ce graphe [8].

Mieux encore, la condition sur le degré des sommets de $G$[5], permet un parcours eulérien *particulier* du graphe $G^*$. En effet, nous avons la proposition suivante :

**Proposition 7.2 :** *le graphe $G^*$ admet un parcours eulérien tel que deux arêtes successives dans ce parcours n'ont jamais la même étiquette.*

**Preuve :** La preuve de cette proposition est constructive. Considérons un sommet quelconque du graphe $G^*$. Un tel sommet est adjacent à $(\delta - 1)$ arêtes

5. A savoir que ce degré est le même pour tous les sommets.

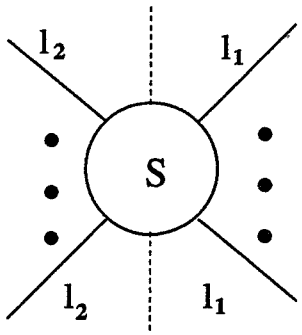


FIG. 7.5 – *Un sommet du graphe $G^*$*

étiquetées par une étiquette $l_1$ et $(\delta - 1)$ arêtes étiquetées par une étiquette $l_2$ (voir figure 7.5). Pour obtenir un parcours eulérien du graphe $G^*$ satisfaisant à la propriété évoquée, on applique l'algorithme distribué décrit dans le chapitre précédent, avec la condition suivante : au niveau d'un sommet $s$, dans la phase de numérotation des canaux, chaque fois que le message SATURER arrive en entrée sur une arête étiquetée $l_1$ (resp. $l_2$), il est envoyé en sortie sur une arête étiquetée $l_2$ (resp. $l_1$). Cette modification est possible puisque, pour l'envoi en sortie du message SATURER, l'algorithme choisit *aléatoirement* le canal de sortie. On obtient alors un parcours eulérien du graphe $G^*$ qui satisfait la propriété mentionnée. □

Sur la base de la proposition 7.2, considérons $(C_+, r)$, un tel parcours de $G^*$ initié à partir d'un sommet racine $r$. $(C_-, r)$ est le parcours du même cycle et à partir de la même racine dans le sens inverse.

A ce niveau, il est important de noter que le problème de l'ordre d'inclusion des arcs du graphe $G^*$ dans le graphe de dépendance $D_F$ est complètement résolu. En effet, les couples $(C_+, r)$ et $(C_-, r)$ permettent d'*ordonner totalement* les arcs de $G^*$. Le principe de l'heuristique que nous proposons consiste alors :

*à partir d'un graphe de dépendance $D_F$ vide, inclure les arcs de $G^*$ suivant l'ordre $(C_+, r)$ puis l'ordre $(C_-, r)$, tant qu'un arc ajouté ne ferme pas un circuit correspondant à un interblocage potentiel dans $G$, avec les arcs déjà inclus.*

En termes de routage, le principe de l'heuristique s'exprime donc comme suit :

*A partir d'un graphe de dépendance $D_F$ vide, inclure les dépendances suivant l'ordre $(C_+, r)$ puis $(C_-, r)$, tant que une dépendance ajoutée n'induit pas un interblocage potentiel dans $G$, avec celles qui sont déjà dans $D_F$.*

Pour chacun des deux parcours, ces dépendances sont interprétées comme sur la figure 7.7, en accord avec le modèle de routage décrit dans le chapitre 4. A

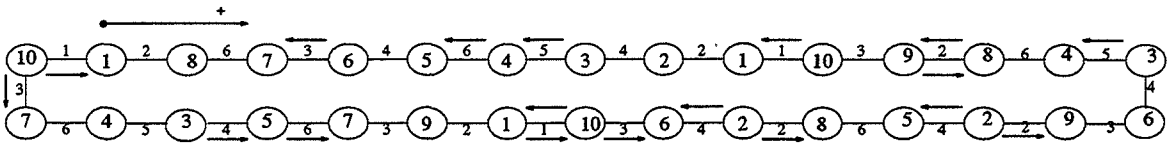

FIG. 7.6 – *Un parcours eulérien de $G^*$*

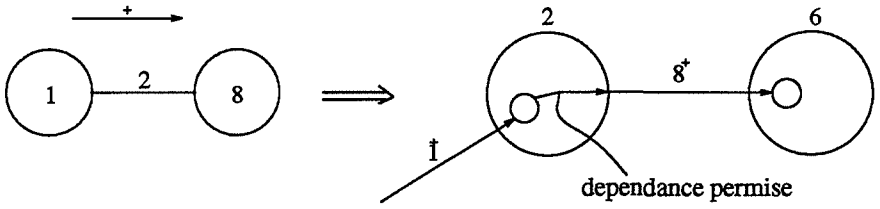


FIG. 7.7 – *Interprétation d'une arête de $G^*$ pendant un parcours*

chaque canal d'entrée est associé un tampon pour le stockage des messages. Ces dépendances sont examinées au fur et à mesure des parcours et suivant qu'elles *ferment* ou pas un *circuit* d'interblocage, elles sont éliminées ou incluses dans le graphe de dépendance $D_F$, et le parcours continue à partir de la dépendance suivante.

La figure 7.6 illustre un parcours eulérien du graphe $G^*$ de la figure 7.2. Le parcours est initié à partir du sommet racine 1. Sur la même figure, nous avons représenté les dépendances éliminées à la suite des deux parcours eulériens. Elles sont marquées par une flèche sur la figure 7.6.

Pour une interprétation de la proposition 7.2, on peut dire que, l'existence d'un parcours eulérien de $G^*$ dont *deux arêtes successives n'ont jamais la même étiquette*, permet d'exhiber une *structure de routage* **cohérente** sur le réseau de graphe $G$. A savoir un *"réseau couvrant"* permettant, un parcours de toutes les dépendances $(a_e, a_s)$[6], qui peut être effectué "**d'un seul coup**" et de manière ininterrompue. Sur ce support de routage, ces dépendances vont être *examinées* suivant l'**ordre** induit par les couples $(C^+, r)$ et $(C^-, r)$.

Le calcul du graphe de dépendance $D_F$, revient au calcul pour tout arc d'entrée $a_e$ du graphe $G$, de l'ensemble:

*$dp(a_e)$*: C'est l'ensemble des arcs de sortie $a_s$ tel que la dépendance $(a_e, a_s)$ est permise,

6. *(arc d'entrée, arc de sortie).*

Dans le but de calculer ce graphe, on associe également à chaque arc d'entrée $a_e$ les deux ensembles suivants :

1. *arcs_père*$(a_e)$ : C'est l'ensemble des arcs $a_p$ tels qu'il existe un chemin de communication du tampon correspondant à $a_p$ au tampon correspondant à $a_e$,

2. *arcs_fils*$(a_e)$ : C'est l'ensemble des arcs $a_f$, tels qu'il existe un chemin de communication du tampon correspondant à $a_e$ au tampon correspondant à $a_f$.

L'algorithme de calcul du graphe de dépendance $D_F$ s'écrit alors comme suit :

DEBUT
POUR (chaque arc d'entrée $a_e$ de $G$) FAIRE
$arcs\_fils(a_e) := \emptyset$; $arcs\_père(a_e) := \emptyset$; $dp(a_e) := \emptyset$;
FINPOUR

*dépendance_courante* := $(r, .)$;
TANTQUE (non fin du parcours $(C_+, r)$ de $G^*$) FAIRE
calculer pour *dépendance_courante* la paire $(a_e, a_s)$ de $G$ correspondante;
SI ($a_e \notin arcs\_fils(a_s)$) ALORS
$dp(a_e) := dp(a_e) \cup \{a_s\}$; $arcs\_fils(a_e) := arcs\_fils(a_e) \cup \{a_s\} \cup arcs\_fils(a_s)$;
$arcs\_père(a_s) := arcs\_père(a_s) \cup \{a_e\} \cup arcs\_père(a_e)$;
FINSI
*dépendance_courante* := *dépendance_suivante*;
FINTANTQUE

*dépendance_courante* := $(., r)$;
TANTQUE (non fin du parcours $(C_-, r)$ de $G^*$) FAIRE
calculer pour *dépendance_courante* la paire $(a_e, a_s)$ correspondante;
SI ($a_e \notin arcs\_fils(a_s)$) ALORS
$dp(a_e) := dp(a_e) \cup \{a_s\}$; $arcs\_fils(a_e) := arcs\_fils(a_e) \cup \{a_s\} \cup arcs\_fils(a_s)$;
$arcs\_père(a_s) := arcs\_père(a_s) \cup \{a_e\} \cup arcs\_père(a_e)$;
FINSI
*dépendance_courante* := *dépendance_suivante*;
FINTANTQUE
FIN

### 7.3.4 Extension de l'heuristique

L'heuristique qui vient d'être décrite est valable uniquement pour des graphes réguliers, où tous les sommets sont de même degré $\delta$. Une telle condition n'est pas très restrictive pour les réseaux d'interconnexion d'architectures parallèles; elle est au contraire souhaitée. En effet, un critère d'efficacité d'un réseau d'interconnexion est la régularité du degré de ces nœuds [25].

Si cette condition sur les sommets de $G$ n'est pas satisfaite, nous proposons la solution suivante, en supposant biensûr qu'il n y a pas une grande disparité entre les degrés des sommets de $G$:

1. Soit $\delta$, le degré maximum des degrés des sommets de $G$,
2. Calculer le graphe $G^*$ de $G$,

3. Rajouter à $G^*$ des arêtes, de telle sorte que tous les sommets est pour degré $2\delta - 1$,

4. Etiqueter les arêtes rajoutées par l'étiquette correspondante,

5. Appliquer la méthode précédente au graphe ainsi obtenu.

Le graphe $G$ de La figure 7.2 illustre ce cas. En effet, les sommets 1 et 5 sont de degré 2, alors que le degré maximum est 4. De ce fait, dans le graphe $G^*$, nous avons introduit deux arêtes, en pointillés sur la figure, entre la paire de nœuds $(1, 10)$ et la paire de nœuds $(3, 4)$. Le nouveau graphe est régulier et tous ses sommets sont de degré 6. La méthode décrite dans la section précédente est ensuite applicable.

Chaque arête rajoutée à $G^*$ est en fait une paire de dépendances virtuelles introduites au niveau de $G$; une dépendance virtuelle par arc rajouté. Si une dépendance virtuelle $(a_e, a_s)$ introduite entre un arc d'entrée $a_e$ et un arc de sortie $a_s$, au niveau d'un sommet $s$, n'est pas éliminée à la suite des deux parcours de $G^*$, cette dépendance correspondra à un tampon de stockage en plus, rajouté à l'arc d'entrée $a_e$, au niveau du sommet $s$. L'arc d'entrée $a_e$ est ainsi divisé en autant d'arcs virtuels que de dépendances rajoutées.

A ce niveau, nous voulons souligner que la technique de la virtualisation est utilisée pour assurer l'existence d'un parcours eulérien de $G^*$ et non pas pour résoudre l'interblocage.

La figure 7.8 représente le graphe de dépendance $D_F$ correspondant au parcours eulérien de la figure 7.6. Sur cette figure, on peut constater les canaux $1_2^-$, $3_2^-$, $1_1^-$, $4_2^+$, $3_1^-$, $4_1^+$, $10_1^-$ et $10_2^-$, induits par le rajout des dépendances virtuelles, en pointillées sur la figure 7.2.

### 7.3.5 Le calcul de la fonction de routage

Au niveau d'un sommet $s$, la fonction de routage est définie suivant le modèle décrit dans le chapitre 4 :

$$\begin{aligned} F_s : A \times S &\longrightarrow \wp(A) \\ (a_e, d) &\longmapsto \{a_s\} \end{aligned}$$

La fonction $F_s$ donne, pour un arc d'entrée $a_e$ d'un message $m$, l'ensemble de tous les arcs de sortie $a_s$ qui se trouvent commencer les *plus courts chemins possibles* vers la destination $d$.

Cette fonction de routage est représentée par une table, $T_s$, à deux entrées, l'arc d'entrée $a_e$ et la destination $d$. Pour calculer cette table, on utilise, comme au chapitre précédent, un algorithme parallèle, inductif dont l'induction porte sur

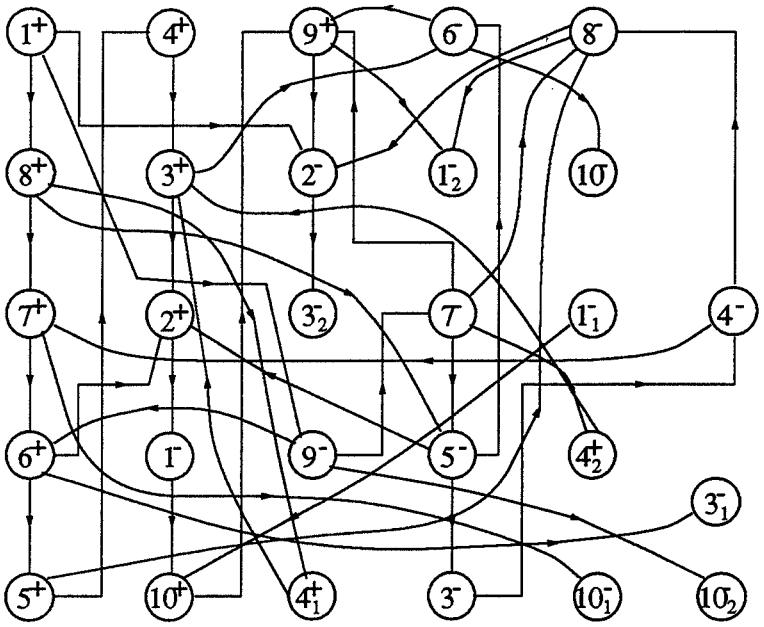


FIG. 7.8 – *Le graphe de dépendance* $D_F$

la longueur des chemins de routage. Cela, afin d'obtenir les chemins de communication les plus courts possibles. Cette induction converge en un nombre de pas d'induction égale au *diamètre* du graphe $G$.

En effet, si l'on pose que :

- $c_s^{(i-1)}(a_e, d)$ : est le coût, en terme de distance, minimum du routage d'un message arrivé à $s$ par $a_e$ et à destination $d$, à la $(i-1)^{ieme}$ itération,
- $T_s^{(i-1)}(a_e, d)$ : est l'ensemble des arcs de sortie du routage d'un message arrivé par $a_e$ à destination $d$, à la $(i-1)^{ieme}$ itération,

nous avons alors les équations suivantes pour le passage de l'itération $(i-1)$ à l'itération $i$ [2] :

$$\begin{cases} c_s^{(i)}(a_e, d) = Min(c_s^{(i-1)}(a_e, d), \underset{(s,v)\in dp(a_e)}{Min}(1 + c_v^{(i-1)}((s, v), d))) & (I) \\ \\ T_s^{(i)}(a_e, d) = \{(s, v)\ \textit{réalisant}\ \ c_s^{(i)}(a_e, d)\} & (II) \end{cases}$$

Cette induction part de l'étape 0 représentant les initialisations adéquates des tables de coût et de routage:

$$(c_s^0(a_e,d), T_s^0(a_e,d)) = \begin{cases} (0,\emptyset) & si\ s = d \\ (1,\{(s,d)\}) & si\ s\ et\ d\ sont\ voisins\ et\ l'arc\ (s,d) \in dp(a_e) \\ (+\infty,\emptyset) & sinon \end{cases}$$

En supposons que:

- $V_s$: représente l'ensemble des voisins du sommets $s$,
- $Calculer\_Nouveaux\_Couts()$: représente la procédure permettant la mise à jour de la table des coûts $c_s(.,.)$, suivant l'équation $(I)$ ci-dessus,
- $Calculer\_Nouveaux\_Liens\_Sortie$: représente la procédure permettant la mise à jour de la table de routage $T_s(.,.)$, suivant l'équation $(II)$ ci-dessus,

le programme parallèle, en $OCCAM$, de calcul des tables de coût et de routage, s'écrit alors comme suit:

```
BEGIN
    SEQ
        FOR (d ∈ S)              /* initialisations */
            FOR (v ∈ V_s)
                c_s^(0)((v,s),d) = { 0  si  s = d
                                   { 1  si  l'arc (s,d) ∈ dp((v,s))   ;
                                   { ∞  sinon

                T_s^(0)((v,s),d) = { ∅        si  s = d
                                   { {(s,d)} si l'arc (s,d) ∈ dp((v,s))
                                   { ∅        sinon

        k := 1;                  /* Phase de calcul */
        WHILE (k ≤ d(G)) DO
            SEQ
                PAR (v ∈ V_s)
                    PAR (v ∈ V_s)
                        (s,v) ! c_s(.,.)
                    PAR (v ∈ V_s)
                        (v,s) ? c_v(.,.)
                Calculer_Nouveaux_Couts();
                Calculer_Nouveaux_Liens_Sortie;
                k := k+1;
END
```

## 7.4 Correction de la méthode

En résumé, on peut dire que la méthode consiste en les étapes suivantes :

1. A partir du graphe $G$, calculer le graphe $G^*$,
2. Calculer un cycle eulérien de $G^*$, tel que deux arêtes successives n'ont jamais la même étiquette,
3. Effectuer un double parcours de ce cycle eulérien, chaque fois qu'un arc ferme un circuit d'interblocage, il est éliminé,
4. Le sous-graphe de $G^*$ résultant est alors le graphe de dépendance $D_F$,
5. Calculer à partir du graphe de dépendance $D_F$, la fonction $F$.

Ainsi, cette méthode de routage prévient l'interblocage par construction. Vis-à-vis de la propriété de complétude, pour *un parcours eulérien quelconque du graphe* $G^*$, la question "reste ouverte". En effet, nous avons "essayé" de montrer la proposition suivante :

**Proposition 7.3 :** *Pour un parcours eulérien quelconque du graphe $G^*$, la fonction $F$ induite par le graphe de dépendance $D_F$, satisfait à la propriété de complétude.*

Cependant, nous n'avons pas de preuve pour la proposition telle qu'elle est formulée ci-dessus. Nous avons également essayé de trouver un contre exemple. En ce sens, nous avons "déroulé" la méthode sur différents graphes réguliers, pour différents parcours eulériens. Pour chaque cas, nous avons toujours trouvé un chemin de communication pour toute paire de sommets.

A défaut d'une preuve de la proposition 7.3, nous proposons la construction ci-dessous, qui assure une fonction de routage valide. Rappelons que le graphe $G$ est régulier de degré $\delta$. Nous distinguons deux cas suivant la parité de $\delta$ :

1. $\delta$ est pair : Le graphe $G$ est eulérien. Le cycle eulérien $(C, r)$ du graphe $G^*$ est alors construit comme suit : *partant de la racine $r$, la première partie de ce parcours correspond à un parcours eulérien $(C', r)$ du graphe $G$.* Ainsi, du graphe $G^*$, le premier arc[7] qui va être éliminé, est l'arc qui ferme le circuit eulérien $(C'_+, r)$. Cet arc a pour extrémité initiale un sommet $s$ et

7. C'est à dire la première dépendance.

pour extrémité finale le sommet $r$. A l'issue des deux parcours eulériens du graphe $G^*$, on distingue deux cas :

(a) Dans le graphe $G^*$, il existe un chemin de $s$ à $r$ : la fonction $F$ est valide, puisqu'il existe un circuit du graphe $G^*$, contenant l'ensemble de ses nœuds,

(b) Dans le graphe $G^*$, il n'existe pas de chemin de $s$ à $r$ : l'arc (s,v) est alors divisé en deux arcs virtuels, dont l'un d'eux est éliminé et l'autre conservé. De même, la fonction $F$ est valide, puisqu'il existe un circuit du graphe $G^*$, contenant l'ensemble de ses nœuds,

2. $\delta$ est impair : Dans ce cas, Le cycle eulérien $(C, r)$ du graphe $G^*$ est construit comme suit : *partant de la racine r, la première partie de ce parcours correspond à un parcours du "Chinese postman" [79] du graphe G.* C'est à dire, un parcours fermé de longueur minimale qui passe par toutes les arêtes de $G$ au moins une fois. On retrouve alors des conditions exactement similaires au cas précédent et le traitement est exactement le même.

   Il est important de noter qu'un tel parcours du graphe $G$ est toujours possible [79]. Cependant, pendant ce parcours, une arête du graphe $G$ peut être visitée dans *le même sens*, plusieurs fois, et des circuits d'interblocage peuvent être induits. Lorsque tel est le cas, un arc est alors divisé en plusieurs arcs virtuels, un pour chaque passage et les arcs qui induisent un interblocage sont éliminés.

La construction ci-dessus implique en fait une propriété plus forte que la complétude. En effet, elle permet un chemin de communication entre toute paire de sommets du graphe $G^*$. C'est-à-dire un chemin de communication entre toute paire d'arêtes du graphe $G$, et par là, plusieurs chemins de communication entre toute paire de sommets de $G$. Ainsi, elle induit une forme d'adaptativité pour le routage dans le graphe $G$.

## 7.5 Application de la méthode aux tores de dimension 2

Dans cette section, nous nous intéressons à une mesure de l'efficacité de la méthode. Pour cela, nous l'appliquons aux grilles toriques de dimension 2 et de base $k$ quelconque.

Nous considérons la même représentation du *tore(2,k)*, que le chapitre 5. En l'occurrence, un système d'axes orthogonaux de cardinal 2, où tout nœud $x$ peut

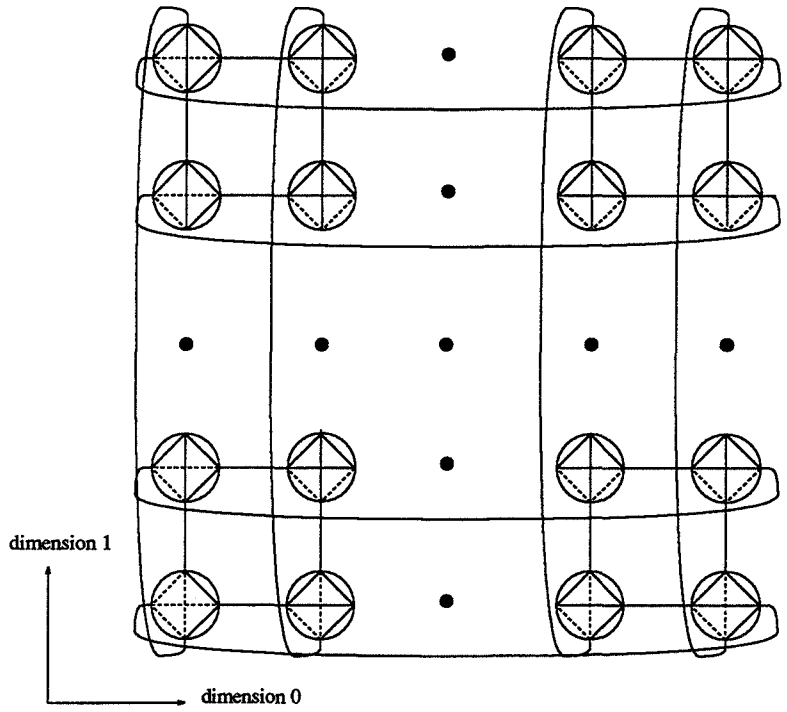


FIG. 7.9 – *Les dépendances induites par la méthode du cycle eulérien*

être représenté par ses 2 coordonnées $(x_0, x_1)$, où $x_i$ est la coordonnée suivant la dimension $i$, pour $i = 0, 1$ et $x_i = 0, ..., k-1$. Le nœud $x$ est connecté à tous les nœuds qui diffèrent sur une seule coordonnée par $(\pm 1 \ mod \ k)$. De même, toute arête peut être repérée par les coordonnées $((x_0, x_1), (y_0, y_1))$ des deux nœuds $x$ et $y$ qu'elle relie.

Pour mesurer l'efficacité de la méthode, nous comparons le nombre de dépendances entre canaux de communication permis par cette méthode, avec celui de la méthode de routage par cycle eulérien, exposée dans le chapitre 5, sur le *tore(2,k)*.

Aussi, pour obtenir une fonction de routage symétrique, i.e. une fonction de routage telle que si $\mu$ est un chemin de communication d'un nœud source $s$ à un nœud destination $d$, le même chemin est également possible de $d$ à $s$, l'heuristique décrite dans la section 7.3 a été légèrement modifiée. En ce sens, nous n'utilisons que le parcours $(C_+, r)$, pendant ce parcours lorsqu'un arc est éliminé (ou permis), l'arc opposé est automatiquement éliminé (ou permis).

La méthode de routage par cycle eulérien permet le même nombre de dépendances quel que soit le cycle eulérien utilisé. Cependant, au niveau d'un sommet

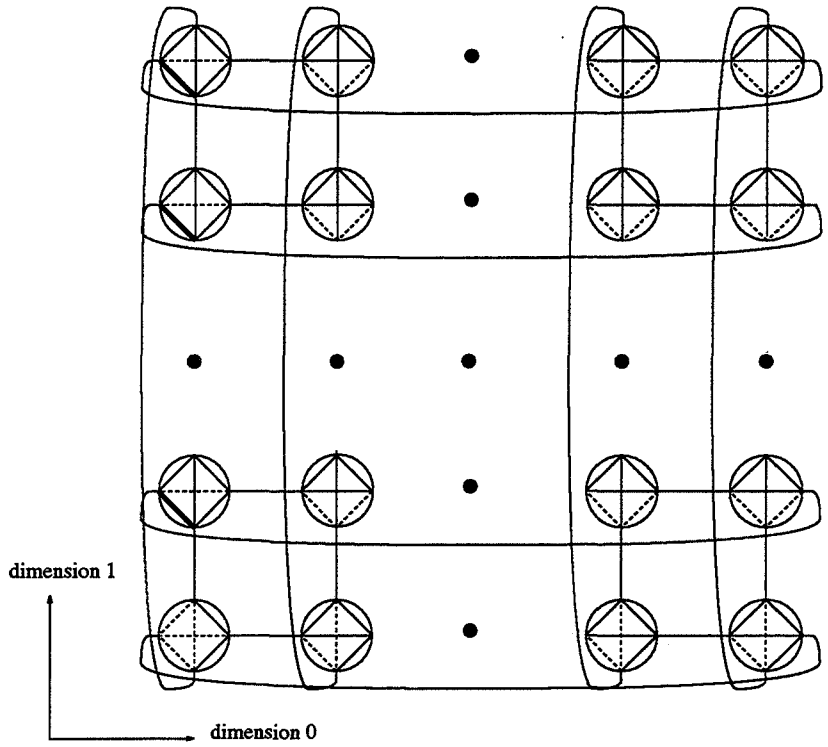


FIG. 7.10 – *Les nouvelles dépendances rajoutées*

quelconque, la répartition de ces dépendances dépend du cycle utilisé. Pour le cycle eulérien décrit à la section 5.5.2 du chapitre 5, sur le tore de dimension 2, les dépendances permises (resp. interdites) sont exactement celles qui sont en trait continu (resp. en pointillés) sur la figure 7.9. Sur cette figure, il n'est pas difficile de voir que le nombre de dépendances permises est :

$$N_{dp} = 2 \times (2 + 4 \times (k^2 - 1)) = 8k^2 - 4.$$

Pour obtenir un nombre de dépendances permises plus grand que le nombre $N_{dp}$, l'essence du problème est de trouver un parcours eulérien du graphe représentatif des sommets du $tore(2, k)$, qui induit un nombre supérieur au nombre $N_{dp}$.

Pour cela, nous remarquons que par rapport à la figure 7.9, lorsqu'on rajoute les dépendances en trait gras sur la figure 7.10, aucune situation d'interblocage potentielle n'est induite par ces nouvelles dépendances rajoutées. Mieux encore, il n'est pas difficile de voir sur la figure 7.10, que ce nouvel ensemble de dépendances

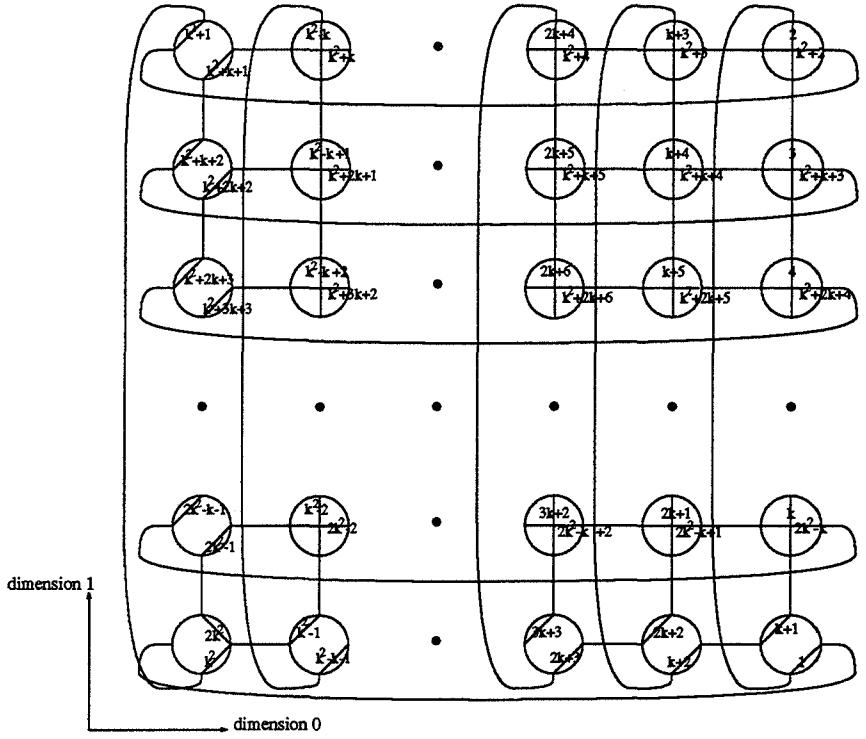


FIG. 7.11 – *La première partie du parcours eulérien du graphe $G^*$*

permises est maximal. En effet, la permission d'une quelconque autre dépendance (en pointillés sur la figure 7.10) induit automatiquement un état d'interblocage avec un sous ensemble de dépendances permises (en traits continus sur la figure 7.10).

Ainsi, notre but est de trouver un parcours eulérien du graphe $G^*$ du $tore(2, k)$, permettant les dépendances représentés sur la figure 7.10.

Par rapport à la construction que nous avons proposée dans la section précédente, la figure 7.11 illustre la première partie du parcours eulérien du graphe $G^*$. L'ensemble des dépendances permises par cette première partie induit une fonction de routage valide.

Nous avons représenté l'ordre d'inclusion des dépendances par un numérotage de celles-ci. Le parcours commence, au niveau du sommet de coordonnées $(k-1, 0)$, par la dépendance entre les deux liens de rebouclage horizontal et vertical. On passe ensuite à la dépendance, au niveau du sommet de coordonnées $(k-1, k-1)$, entre le lien de rebouclage vertical et le lien $((k-1, k-1), (k-1, k-2))$, et ainsi de suite.

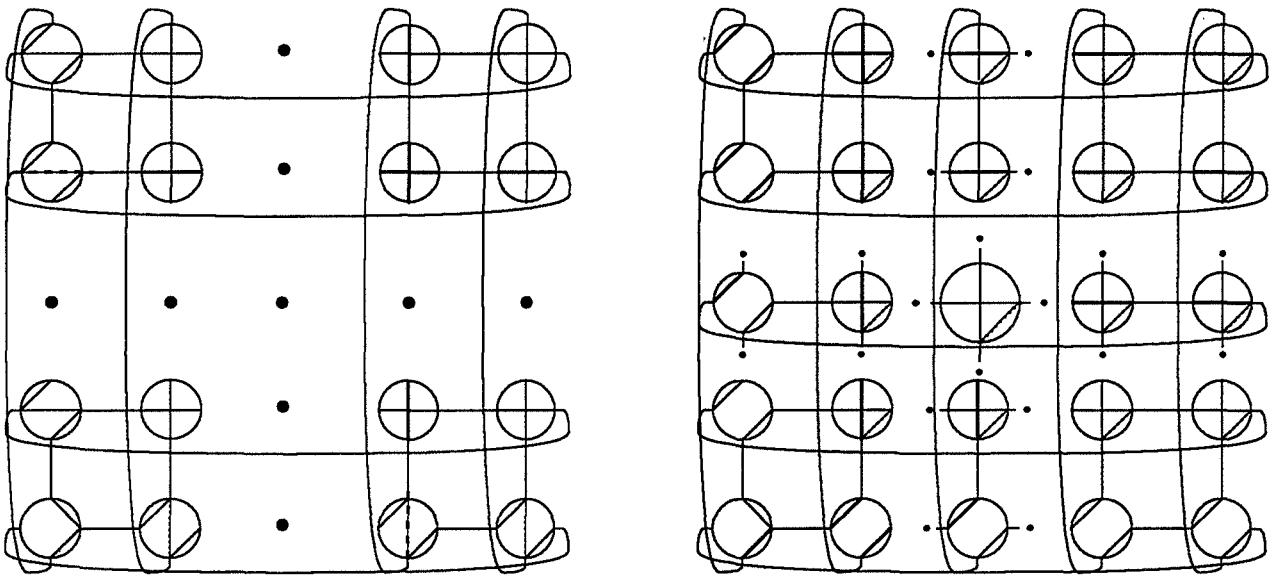


FIG. 7.12 – *Les cas* 1 *et* 2

Cette première partie possède une caractéristique particulière. En effet, les dépendances permises sur la figure 7.11 impliquent, *" de facto"* par elles mêmes, que quelle que soit la suite du parcours eulérien du graphe $G^*$, l'ensemble des dépendances permises (resp. interdites) obtenues à la fin du parcours, sont exactement les dépendances permises (resp. interdites) de la figure 7.10.

Pour montrer cela, il suffit de montrer que pour chaque dépendance interdite de la figure 7.10, il existe un ensemble de dépendances de la figure 7.11 avec lequel elle induit un interblocage.

En prenant comme convention de repèrer une dépendance par les deux arêtes qu'elle relie, on peut optimiser la preuve et distinguer en fait huit cas correspondant à différents types de dépendances :

1. Les dépendances de la forme $\{((x_1, y_1), (x_1, y_1 + 1)), ((x_1, y_1 + 1), (x_1 + 1 \ mod \ k, y_1 + 1))\}$ : ce cas est représenté sur la figure 7.12(b). Nous avons choisi une dépendance de ce type[8] et nous avons représenté[9] le sous-ensemble de dépendances permises de la figure 7.11 avec lequel elle induit un interblocage. Toutes les dépendances de la forme précédente induisent

8. En pointillés dans un cercle plus grand que les autres.
9. En trait gras.

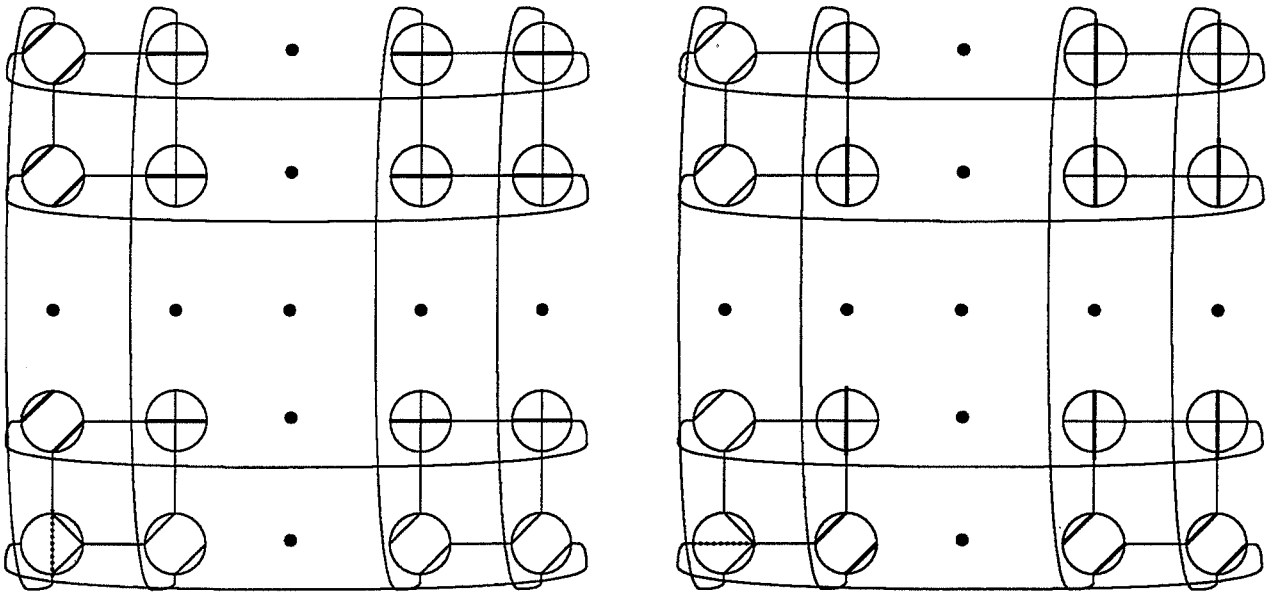


FIG. 7.13 – *Les cas* 3 *et* 4

un interblocage avec un "parcours" du même style que celui représenté sur la figure 7.12(b) en trait gras.

2. Les dépendances de la forme $\{((x_1, k-1),(x_1,0)),((x_1,0),(x_1,1))\}$ ou $\{((k-1,y_1),(0,y_1)),((0,y_1),(1,y_1))\}$ : la figure 7.12(a) concerne ce type de dépendances. De même, nous avons choisi deux dépendances interdites[10] et nous avons représenté pour chacune, le sous ensemble de dépendances permises[11] de la figure 7.11, avec lequel elle induit un interblocage.

3. **La** dépendance $\{((0,1),(0,0)),((0,0),(0,k-1))\}$ : la figure 7.13(a) illustre cette dépendance[12] et l'ensemble des dépendances permises[13] avec lequel elle induit un interblocage.

4. **La** dépendance $\{((k-1,0),(0,0)),((0,0),(1,0))\}$ : la figure 7.13(b) montre le parcours suivant lequel, lorsque cette dépendance est permise, elle induit un état d'interblocage potentiel.

10. Une pour chaque type, représentée en pointillés gras.
11. Représentés en trait continu gras.
12. En pointillés.
13. En trait gras.

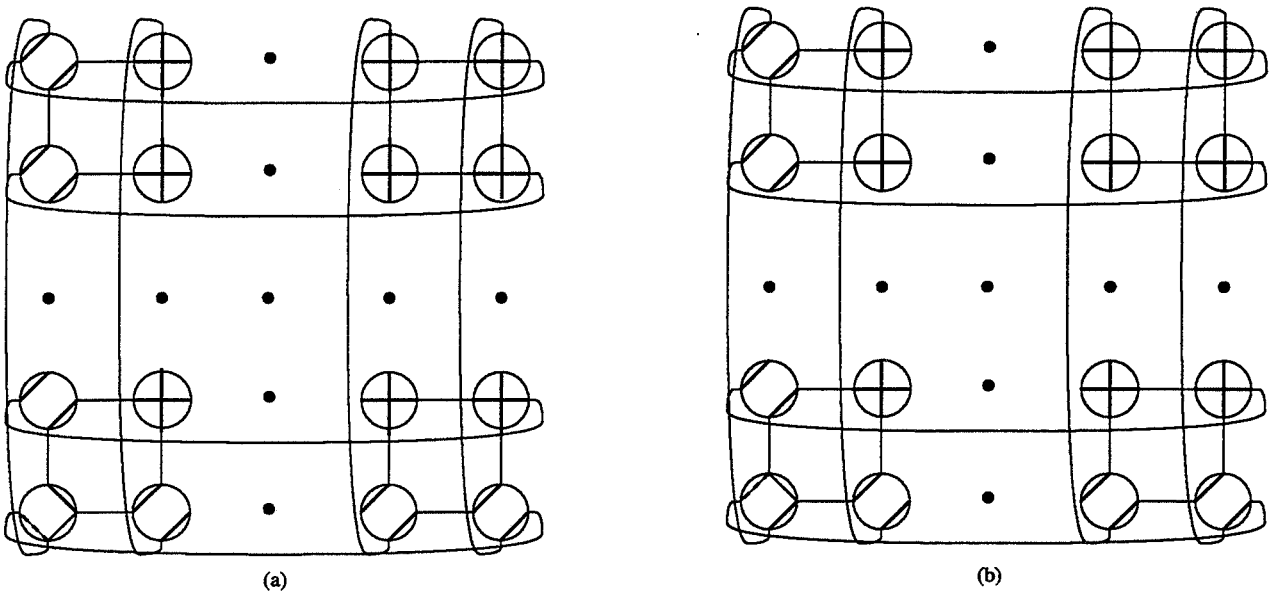


FIG. 7.14 – *Les cas* 5 *et* 6

5. **La** dépendance $\{((0,k-1),(0,0)),((0,0),(k-1,0))\}$ : la figure 7.14(a) montre cette dépendance ainsi que le parcours suivant lequel elle induit un interblocage.

6. **La** dépendance $\{((k-1,0),(0,0)),((0,0),(0,1))\}$ : ce cas est illustré sur la figure 7.14(b).

7. Les dépendances de la forme $\{((x_1,0),(x_1+1,0)),((x_1+1,0),(x_1+1,k-1))\}$ : la figure 7.15(a) concerne ce cas. Nous avons choisi une dépendance interdite[14] de ce type et nous avons représenté, le sous ensemble des dépendances permises[15] de la figure 7.11, avec lequel elle induit un interblocage.

8. Les dépendances de la forme $\{((x_1,y_1),(x_1+1,y_1)),((x_1+1,y_1),(x_1-1,y_1-1))\}$ : la figure 7.15(b) montre une dépendance de ce type, et le sous ensemble de dépendances permises avec lequel, elle induit un interblocage.

Ainsi, nous avons exhibé pour chaque dépendance interdite sur la figure 7.10, un ensemble de dépendances permises de la figure 7.11, avec lequel elle induit un interblocage. Ceci montre que les dépendances permises par la première partie du

14. Représentée en pointillés gras.
15. Représentées en trait continu gras.

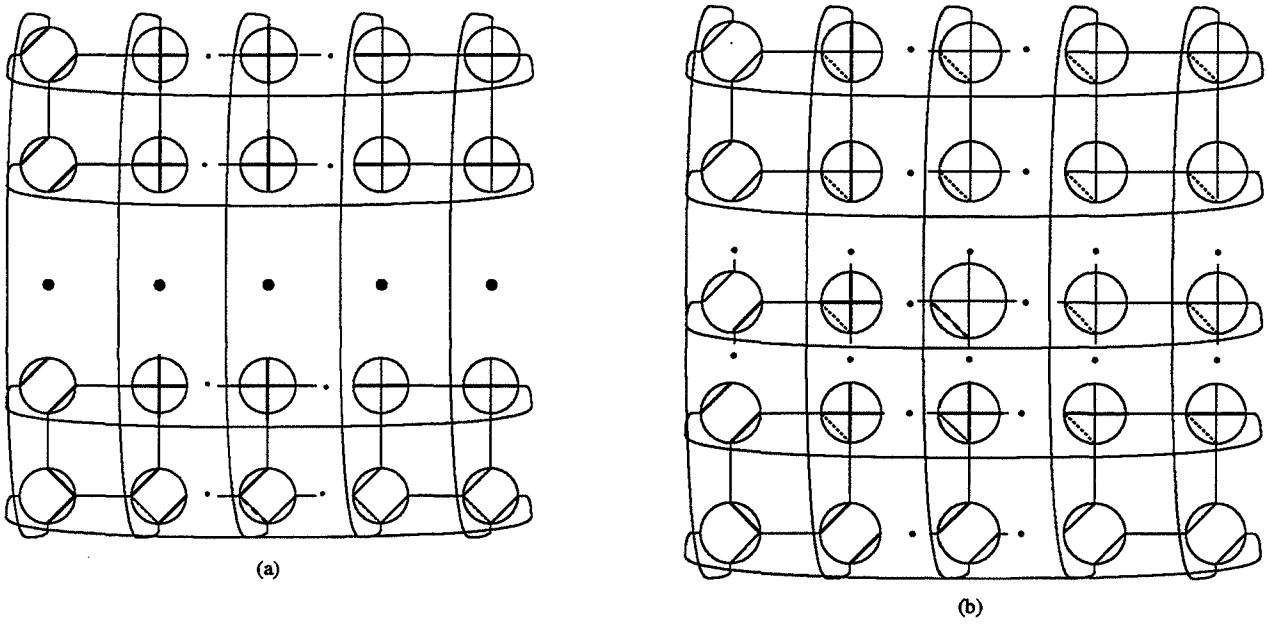


FIG. 7.15 - *Les cas 7 et 8*

parcours eulérien de $G^*$, de la figure 7.11, induisent d'elles mêmes, les dépendances représentées sur la figure 7.10. Ainsi, le parcours eulérien de $G^*$ peut être continué de manière *aléatoire* à partir de la dépendance numérotée $2k^2$ sur la figure 7.11 et le résultat sera exactement les dépendances de la figure 7.10.

Pour les *tore(2,k)*, le parcours eulérien exhibé du graphe $G^*$ permet d'énoncer les deux conclusions suivantes :

1. Le nombre de dépendances permises est $8k^2 - 4 + 2(k-1) = 8k^2 + 2k - 6$, et est supérieur à celui de la méthode de routage par cycle eulérien,

2. Les résultats donnés au chapitre 5, concernant le nombre de niveaux nécessaires pour un facteur d'élongation unitaire, sont également valables lorsque le schéma $SCP$ est la méthode de routage décrite dans ce chapitre.

## 7.6 Conclusion

Dans ce chapitre, nous avons présenté une méthode de construction de fonctions de routage sans interblocage dans les réseaux de processeurs. Cette méthode est assez générale pour être applicable à une large classe de réseaux d'interconnexion. Elle utilise un espace mémoire indépendant de la taille du réseau pour le

stockage des messages utilisateurs, en l'occurrence un tampon par canal de communication. Elle route dans certains cas sur les plus courts chemins du réseau.

Cette méthode part d'une *spécification formelle* du problème du routage sans interblocage dans un réseau de processeurs. A savoir, *l'élimination d'un ensemble de dépendance dans le graphe $G^*$ qui élimine tous les circuits d'interblocage dans $G$.*

Notons toutefois que le choix du cycle eulérien utilisé pour le parcours de l'ensemble des dépendances, ainsi que le choix du sommet racine et du premier sens de parcours, ont une incidence directe sur la qualité du schéma de communication final dérivé, en particulier sur le nombre total de dépendances permises.

Pour un réseau d'interconnexion donné, le choix de ces éléments peut être guidé par des critères spécifiques à la topologie du réseau. L'application aux tores de dimension 2 en est une illustration.

La méthode qui a été décrite dans ce chapitre est proposée comme un schéma $SCP$, pour une stratégie de routage à base de la communication multi-niveaux. En effet, comme nous l'avons mentionné dans le section introductive, le schéma primaire a la priorité de résoudre efficacement le problème de l'interblocage, tout en assurant la propriété de complétude et l'utilisation de tous les liens de communication. La méthode de dérivation de fonction de routage qui vient d'être exposée, assure l'ensemble de ces critères, tout en essayant de maximiser le nombre de dépendances à permettre. Ce dernier aspect contribue alors à ce que le nombre de niveaux de communication nécessaire à l'intégration des critères d'efficacité soit minimum.

# Conclusion générale et perspectives

Dans cette thèse, nous nous sommes intéressés au problème du routage correct et efficace, basé sur un mode de communication de type *(un émetteur, un récepteur)*, dans les architectures massivement parallèles à mémoire distribuée.

## Bilan de la contribution

Au **chapitre 2 de la première partie**, nous avons présenté un état de l'art sur les problèmes du routage dans ce type d'architecture. Une classification des concepts liés au routage a été donnée. La problématique du routage dans les calculateurs parallèles à mémoire distribuée a été dégagée.

Parmi les différents éléments de cette problématique, **au chapitre 3 de la première partie**, nous avons focalisé notre attention sur un aspect particulier : *la stratégie de routage*. En effet, celle-ci constitue un élément principal dans la réalisation de noyaux de routage.

La **deuxième partie** constitue l'apport de cette thèse. En l'occurrence, une méthode générale de conception de stratégies de routage : *la communication multi-niveaux* et le *schéma de communication primaire* associé.

La démarche globale de cette méthode est comme suit :

1. Le schéma $SCP$, proposé au **chapitre 7**, cible spécifiquement le problème de l'interblocage. Ce schéma prévient ce problème d'une manière efficace. C'est en quelque sorte une structure de routage "*plate*" sur le réseau d'interconnexion. Par ailleurs, il utilise tous les liens de communication et satisfait à la propriété de complétude. Ainsi les critères de correction sont assurés par le schéma de communication primaire.

   D'un autre côté, nous avons constaté que la notion de dépendance entre deux canaux de communication successifs était essentielle pour le problème du routage dans les réseaux de processeurs. Plus grand est le nombre de

dépendances permises, plus la fonction de routage peut intégrer potentiellement des critères d'efficacité. A ce niveau, il est également important de noter que la répartition des dépendances permises, à travers le réseau de communication, influe également sur la qualité de la fonction de routage. Ainsi, le schéma $SCP$ que nous avons décrit essaye de maximiser le nombre de dépendances à permettre, tout en assurant une répartition assez uniforme de ces dépendances sur le réseau.

2. Pour intégrer les critères d'efficacité restants; i.e. principalement, un routage pour tout couple de nœuds sur plusieurs chemins minimaux, une forme de virtualisation est nécessaire. Nous avons proposé, au **chapitre** 5, la communication multi-niveaux. En l'occurrence, une hiérarchisation du schéma primaire en niveaux virtuels de routage, telle qu'une transition de niveau corresponde à l'amélioration de l'efficacité d'un chemin de communication. De là, découle le principe incrémental de la méthode. En effet, on utilise la technique de la virtualisation dans le but d'améliorer le schéma primaire et par une démarche *"globale"*.

3. La paire (schéma $SCP$, communication multi-niveaux) résulte en un schéma $SCG$ du réseau d'interconnexion qui satisfait d'une part, les critères de correction et d'autre part, les critères d'efficacité en fonction du nombre de niveaux. Pour calculer les chemins de communication, la méthode que nous avons proposée utilise ensuite une technique de programmation dynamique permettant le calcul des tables de routage. Notons également que la méthode que nous avons proposée pour l'implantation de cette technique permet d'une part, de calculer les tables de routage directement sur le réseau de processeurs, et d'autre part, de remédier au problème de l'attraction des niveaux de communication supérieurs.

Le problème qu'induit cette méthode est qu'elle utilise systématiquement des tables de routage de taille $O(N)$, où $N$ est le nombre de nœuds du réseau. C'est-à-dire une représentation classique, dépendant de la taille du réseau, et non *compactée*, de la fonction de routage. Cependant, une étude sur la complexité mémoire du routage dans les réseaux de processeurs [38], montre qu'*asymptotiquement* la représentation classique, sous forme de tables à deux entrées est la méthode *générale*, la plus compacte.

Pour un réseau d'interconnexion donné, le point important de cette méthode, est de déterminer un "*meilleur*" schéma $SCP$, de sorte à rendre le nombre de niveaux de communication nécessaires à l'intégration des critères d'efficacité, indépendant de la taille du réseau.

Bien que le principe de la communication multi-niveaux ait été validé[16] en utilisant deux schémas $SCP$ autres que celui décrit au chapitre 7, ce dernier est le schéma $SCP$ adéquat. L'application aux tores de dimension 2 illustre cela. En effet, il part d'une spécification formelle du problème du routage sans interblocage dans un réseau de processeurs. Ce schéma permet ainsi de retrouver les deux schémas $SCP$ du chapitre 5. Il suffit pour cela d'adapter l'heuristique de calcul du graphe de dépendance.

Au niveau réalisation, **au chapitre 6**, nous avons décrit une implantation de cette méthode, sur un réseau de TRANSPUTERS T805, lorsque le schéma $SCP$ est le routage par cycle eulérien. Pour obtenir un système auto-constructif, nous avons également décrit, **au chapitre 6**, un algorithme distribué de calcul d'un tel cycle dans un réseau. Cet algorithme utilise un nombre de messages égal à deux fois le nombre de liens de communication du réseau. Il n'est donc pas très coûteux en nombre de messages générés.

Les résultats de simulation effectués sur les grilles toriques, nous ont permis d'exhiber un parcours eulérien particulier de ce réseau, tel que l'application de la méthode sur ce parcours résulte en un algorithme de routage minimal, sans interblocage, adaptatif, qui utilise $d$ niveaux de communication et une seule table de routage de taille $O(k^d)$, pour une grille torique de dimension $d$ et de base $k$ (**chapitre** 5).

Au travers d'un autre parcours eulérien de la grille torique, nous avons amélioré ce résultat en un premier algorithme de routage minimal, sans interblocage, adaptatif, qui utilise 2 niveaux de communication et une seule table de routage de taille $O(k^d)$. A partir de là, nous avons déduit un second algorithme de routage dans les grilles toriques multi-dimensionnelles, minimal, sans interblocage, qui utilise 2 niveaux de communication et un espace mémoire constant pour la représentation de l'information de routage [50]*. Cet algorithme n'est cependant pas adaptatif (**chapitre** 5).

Une autre application de la communication multi-niveaux aux grilles toriques multi-dimensionnelles a été étudiée, **au chapitre** 5, en utilisant un schéma de communication primaire de type *e-cube*. L'algorithme de routage obtenu est minimal, sans interblocage, utilise un espace mémoire constant pour la représentation de l'information de routage et $d$ niveaux de communication, pour une grille torique de dimension $d$ [52]*.

16. Ceci en raison de l'implantation sur le réseau de TRANSPUTERS que nous avons dévellopé.

# Perspectives

Les travaux réalisés dans le cadre de cette thèse, ouvrent de nombreuses perspectives. Parmi celles-ci, on peut noter les points suivants :

1. L'algorithme de routage dans les grilles toriques décrit dans [50]* est minimal, sans interblocage, utilise 2 niveaux de communication et un espace mémoire constant pour la représentation de l'information de routage. Cependant, cet algorithme ne permet qu'un seul chemin de communication entre toute paire de nœuds.

   Il est évident qu'une direction d'amélioration de cet algorithme doit assurer l'adaptativité, tout en gardant un espace mémoire constant pour la représentation de la fonction de routage. Aussi, il est évident que le *"degré d'adaptativité"* dépend du nombre de niveaux de communication.

   Ainsi, une amélioration de cet algorithme consiste à déterminer la relation arithmétique qui existe entre les coordonnées d'un nœud et le numérotage (suivant le parcours eulérien exhibé) de ses liens adjacents. Cette relation arithmétique permettra alors, d'une part, une représentation de la fonction de routage indépendante de la taille du réseau, et d'autre part, de déterminer quand un message peut être envoyé sur un lien autre que celui spécifié par l'algorithme. Cela, étant bien entendu dans le but d'assurer l'adaptativité.

2. Le deuxième point concerne le schéma $SCP$ que nous avons proposé. Ce nouveau schéma ouvre plusieurs directions quant à la dérivation de fonctions de routage sans interblocage dans les réseaux de processeurs. Cela, principalement à cause du fait qu'il part d'une *spécification formelle* du problème du *routage sans interblocage* dans les réseaux de processeurs.

   Parmi les directions possibles, pour un réseau d'interconnexion donné, la détermination d'un "meilleur" parcours eulérien du graphe adjoint peut être guidé par des aspects liés à la topologie du réseau. Aussi, une fois un tel parcours fixé, il serait intéressant d'essayer de représenter au niveau d'un nœud, la fonction de routage par une relation arithmétique reliant les coordonnées du nœud et "un numérotage" de ses liens de communication adjacents. Cela permettrait d'une part, une représentation de la fonction de routage indépendante de la taille du réseau et ainsi de remédier au problème qu'induit la méthode que nous proposons, et d'autre part, la détermination (de manière locale au niveau d'un nœud) de la politique qui définit les transitions de niveau, en fonction justement du nombre de niveaux.

# Troisième partie

# Annexe

# Annexe A

# The Multi-Level Communication : Minimal, Deadlock Free, Storage Optimal Routing for Torus Networks [50]

B.HADIM I.SAKHO
Centre SIMADE - Ecole des Mines de St Etienne
158 Cours Fauriel 42023 St Etienne Cedex 2 France
e-mail : {sakho,hadim}@emse.fr - tel : 77 42 01 72 - fax : 77 42 66 66

Abstract

In this paper, we present a general methodology for the improvement of the criteria of a given routing scheme. We study particularly the correlation between deadlock-avoidance and paths' stretch factor. We prove that the methodology conserves deadlock-avoidance. As the methodology starts from a *primary communication scheme*, we choose routing by Eulerian cycle rules [73] as the primary scheme. We apply the resulting routing scheme to the torus network, and prove that this result on an efficient routing algorithm for such networks. [1]

## A.1 Introduction

Massively parallel distributed memory MIMD computers speedup applications execution by allowing *communicating processes* to run *concurrently* on different processors. In this type of architectures, processors are interconnected by a set of

1. Ce papier est disponible en rapport interne *RR. 97.5* du Centre SIMADE-ENSMSE.

*point to point* bidirectional communication links, according to a certain *topology*, and communicate by *exchanging messages* through these links. As in general, communicating processes are not always located in neighboring processors, one of the fundamental problems for making use of applications efficient is **communications** between distant processes. Thus, a specific *communication kernel* is required. A principal function of such a kernel is to virtualize the processors interconnection network in a fully connected one, in order allowing message exchanges between any pair of processors. This requires constructing paths within the interconnection network through which messages are routed. For this purpose, every path-constructing *strategy must* integrate a set of **correction** and **efficiency routing criteria**. A correction criterion is a constraint such strategy should *necessarily* satisfy. An efficiency criterion is a desirable constraint to be satisfied. Among the set of correction criteria, we can cite:

1. **Completeness**: the strategy must built at least one path for each pair of (source,destination) processors.

2. **Deadlock avoidance**: the *routing scheme* derived from the strategy must be deadlock free.

Efficiency criteria are essentially:

1. **Constant memory space**: the memory space allocated either for message storing buffers or representing the routing scheme, has to be network size *independent* (with regard to the number of processors).

2. **Minimal paths**: the strategy must build, for each pair of (source,destination) nodes, minimal routing paths.

3. **Use of all communication links**: the strategy has to use all communication links in the network and balance the communication rate fairly over these links.

4. **Several paths**: it is also preferable to have several routing paths between each pair of nodes; so to allow an *adaptive routing scheme* which avoids *local congestion* and tolerate *physical faults.*

Among correction criteria, deadlock avoidance is as important as completeness. Indeed, in massively parallel architectures, the communication rate between distant processes is so high that it favors the occurrence of deadlock situations. Therefor, it is essential for a communication kernel to be deadlock free.

Deadlock is a situation where a set of $n$ processors develops a *dependency loop*: each processor $p_i$ attempts to communicate with its neighbor in the loop

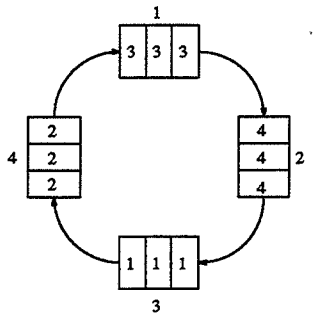


FIG. A.1 – *A Deadlocked State*

$p_{(i+1) \bmod n}$ and can not because this latter has all its resources (messages storing buffers or communication channels) busy (cf. fig1) [23]. Deadlocks can be solved by *restricting* the use of communication links [52]. Such restrictions can result in *stretching* the routing paths length with respect to the optimal ones [52]. On the other hand, the *length* of this routing paths affects directly the performance of the communications. The shorter these paths are, the better the communication delay is. Moreover, note that this criteria has already been used to measure the efficiency of a routing strategy. That is, in [78], UPFAL and PELEG define the *paths' stretch factor* of a routing strategy as the maximum ratio, over all pairs of nodes, between the length of a path produced by the strategy and the length of a minimal path between the same pair of nodes. Unfortunately, minimal paths routing supposes no restrictions between incoming and outgoing links of a processor. This lack of restrictions can result on deadlocks situations. Consequently, optimal routing and deadlock avoidance are *conflicting* criteria. Satisfying them simultaneously may requires per processor a large memory amount for storing messages or representing the routing informations [78].

In this paper, we are interested in a *good correlation* between deadlock avoidance (as a correction criterion) and paths' stretch factor (as an efficiency criterion) while minimizing the memory space either for storing users' messages or representing the routing informations. In other words, we would like to devise a constructing methodology of *routing strategies* which are deadlock free and allow as good paths' stretch factor as desirable while requiring (for some regular networks) network *independent* memory space.

Few research has been done according to this optic. Indeed the literature has focused either on solving the deadlock problem by proposing routing strategies which do not take into account plainly the paths' stretch factor [14, 23, 24, 28, 39, 40, 43, 45, 46, 72, 80]; or on defining routing strategies, not systematically

deadlock-free, which allow a good *tradeoff* between the paths' stretch factor and the memory space necessary for representing the routing strategy [5, 78]. Two thought schools prevail in the solutions of deadlock:

- **Detection-Recovery methods:** this approach proceeds by detecting the deadlock situation, when it occurs, and then starts a recovery mechanism to resolve it [14]. It is obviously inadequate in massively parallel context.

- **Prevention methods:** these methods *prevent* a deadlock situation while designing the communication kernel. Thus, it can never occur. This is the solution adopted for massively parallel architectures [3, 23, 24, 28, 39, 40, 43, 45, 46, 67, 72, 80].

Prevention methods rely on total ordering techniques of communication resources. First solutions suggested in the literature define an order on the message storing buffers. Based on the *necessary* and *sufficient* conditions previously cited, an *acyclic routing structure* is constructed on the communication network, by using as many message buffers as needed and routing is performed according to rules [3, 40, 43, 45, 46]. These methods require memory space depending on the network *diameter*. They are inadequate for massively parallel architectures. In [23, 24], DALLY and SEITZ have suggested an original approach with respect to this problem and provide an *a priori* characterization for the existence of a deadlock state. This characterization is based on the notion of *dependency graph*. They prove that given an *interconnection network I, a routing function R on I, is deadlock free if and only if its channel dependency graph does not contain cycles.* On the Basis of this theorem, they gave a methodology for constructing deadlock-free routing function using *virtual channels*. DALLY&SEITZ theorem provides a good approach to prevent deadlocks. Furthermore, several recent solutions are based on it [67, 44, 72]. In [28], DUATO observes that DALLY&SEITZ' necessary and sufficient condition is valid only for deterministic routing functions which define a unique path for each (*source, destination*) pair of nodes. He proves that for adaptive routing functions , deadlocks *can* be avoided even when the dependency graph contains cycles. It just *suffices* to *choose* paths that are not in these cycles. Based on this observation, he gives a methodology for constructing deadlock-free *adaptive* routing functions. In fact, a cycle in the channel dependency graph just implies a *potential* deadlocked state in the network, which is not *necessary reachable*. The real occurrence of the deadlocked state depends on the *dynamical* behavior of the communication patterns. Thereby, constructing routing functions whose dependency graph is acyclic, is a *strong* condition to prevent deadlock. Nevertheless, this condition may be *necessary* when considering other criteria that a routing strategy must satisfy, such as *fault tolerance* and *congestion avoidance*.

These two criteria *restrict* the number of routing paths when they are not satisfied. The *remaining paths* must *preserve the deadlock-free property*, in order to ensure an *efficient* and *robust* routing kernel. In summary, *all* routing paths must ensure deadlock avoidance.

The problem of efficient routing (in terms of paths' stretch factor) with regard to the necessary memory space to represent a routing strategy was raised in [63] for large interconnection computer networks. In [78], the problem was dealt with in a general way for interconnection networks. Rather than designing a scheme for some fixed stretch factors, the method is parameterized by the entire range of possible stretch factors. In particular, the hierarchical routing schemes presented guarantee a stretch factor of $12k + 3$ while requiring a total of $O(k^3 n^{1+1/k} logn)$ bits of memory space at each node (for every $k \geq 1$). In [5] AWERBUCH&*al* propose several routing methods for general networks which attempt to overcome some problems mentioned about the methods [78]. Still, all these routing schemes do not take, in their formulation, the deadlock problem explicitly into account. In [37], GAVISH&*al* study the problem of minimal buffers number required to avoid *store* and *forward* deadlock. They state the problem as a multi-commodity flow one with integer constraints and give some heuristics for its solution. But, their formulation do not take in account the routing paths length. These are not guaranteed to be minimal.

In this paper, we propose a constructing methodology of routing schemes which allows at the same time deadlock avoidance and regulation of paths' stretch factor: the `multi-level communication`. For some *regular* networks like the torus, the memory space induced by this methodology is network-size independent; when a judicious choice is made with respect to certain parameters. In section 2 we give the principle of the methodology, with an application to the pair $(deadlock - avoidance, paths'\ stretch\ factor)$. We prove deadlock-avoidance for the resulting routing scheme. In section 3 we apply the methodology, based on Eulerian cycle routing rules [72], to the torus network. We prove that with two communication levels, minimal routing is obtained. Then, we give a minimal, deadlock-free, optimal storage, deterministic routing algorithm for such networks. In section 4, we discuss several points related to the contains of the paper.

## A.2 The multi-level communication

We begin by giving the multi-level communication principle for a general $(correction, efficiency)$ pair of criteria, in an abstract way. Then we apply this methodology to deadlock avoidance and paths' stretch factor.

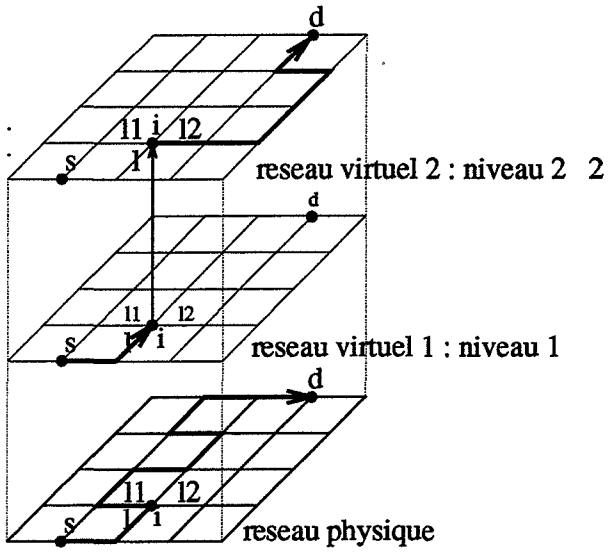


FIG. A.2 – *Multi-Level Communication Principle*

### A.2.1 Principle

Multi-level communication is based on a *communication level* idea. A communication level is a *virtual structure of routing* induced by a **primary communication scheme**. The principle of the multi-level communication consists in associating several levels of communication with this communication scheme. These levels are *ordered* in an obvious way from level 1 to the upper one. At the same level, routing is carried out according to the primary communication scheme and moves from level $i$ to level $i+1$ each time we need to improve the efficiency of a routing path.

For a processor interconnection network $I$, assume that the correction criterion is induced by a set of *formal routing rules* $\mathcal{R}$ on $I$; we call $\mathcal{R}$ the primary communication scheme. On the other hand, assume that the efficiency criterion is determined by a *pattern* $\mathcal{P}$ which defines, for each pair of $(source, destination)$ nodes, at least an efficient routing path. Multi-level communication consists in routing, at the same level, according to the primary communication scheme $\mathcal{R}$ as long as the routing path followed matches the routing path induced by the pattern $\mathcal{P}$. At a given node $n$, the routing scheme moves from level $l$ to level $(l+1)$ all times it *need* to infringe a rule of $\mathcal{R}$, in case the induced remaining routing path does not match a path induced by $\mathcal{P}$, and for the purpose of ensuring the efficiency criteria. At level $(l+1)$, routing resumes according to $\mathcal{R}$ following the newly chosen path. This principle is applied at each node, each time we need to

ensure matching between the paths induced by $\mathcal{R}$ and $\mathcal{P}$, as long as the upper level has not been reached.

The communication levels may be of different nature depending on the scheme $\mathcal{R}$ nature (virtual channels, virtual networks, etc). They exhibit a certain form of *replication* using the general concept of *virtualization.* Schematically, these levels may be *represented* as a *superposition* of *virtual networks* identical to the physical one (cf. fig. 2). In figure 2, assume that $m$ is a message to send from node $s$ to node $d$ following the path in bold-line, according to $\mathcal{R}$. Then:

- initiating routing of message $m$ at virtual network 1 from $s$,
- until node $i$ the scheme $\mathcal{R}$ and the pattern $\mathcal{P}$ induces the same routing path,
- at node $i$, the output link induced by $\mathcal{R}$ (respectively by $\mathcal{P}$) is $l1$ (respectively 12), so there is no match,
- then message $m$ is sent through link $l_2$ of virtual network 2; i.e. we infringe a rule of $\mathcal{R}$ and go up by a communication level,
- message $m$ is then routed in virtual network 2 according to $\mathcal{R}$ until it reaches its destination (we suppose the scheme $\mathcal{R}$ and the pattern $\mathcal{P}$ induce the same routing path (in bold-face) for the pair $(i, d)$).

Thereby, the correction (resp. efficiency) criterion has been respected, since message $m$ has been routed from $s$ to $d$ according to $\mathcal{R}$ (resp. $\mathcal{P}$), in two phases.

Now, assume that $\mathcal{R}$ is a set of deadlock free routing rules on the processors interconnection network $I$. Assume that $\mathcal{P}$ is the communication scheme without *constraints*; i.e. the scheme which routes, for each (*source*, *destination*) pair of nodes over the network *minimal* paths. $\mathcal{P}$ represents the *transitive closure* of the graph associated with $I$. In general, at a processor $p$, for a fixed destination $d$, a communication scheme can be *viewed* as a set of couples of links $(l_i, l_o)$, $l_i$ (resp. $l_o$) standing for input (resp. output) link. This means that a message $m$ entering $p$ by $l_i$ to destination $d$ is sent through $l_o$. Thus, for all $I'$ destinations, this communication scheme may be viewed as a set of the same form. Following this discussion, the scheme $\mathcal{R}$ does not *certainly* include *all* the pairs $(l_i, l_o)$ belonging to the scheme $\mathcal{P}$. Thus, all $\mathcal{R}'$ routing paths between two nodes $(s, d)$, including a pair of links $(l_i, l_o)$ which is not in the scheme $\mathcal{P}$, are in fact *stretched* paths with regard to the minimal ones. Then, assume a superposition of $l$ virtual networks $vn_i, i = 1 \ldots l$, identical to the physical one. Routing in each virtual network is carried out according to the scheme $\mathcal{R}$. Let $m$ be a message in $vn_i$ of level $i$. Suppose that at processor $p$, message $m$ arrived by input link $l_i$ is to be sent through output link $l_{o_1}$ to destination $d$ according to the scheme $\mathcal{R}$. Assume that

the pair $(l_i, l_{o_1})$ is not in the scheme $\mathcal{P}$. Suppose also that at processor $p$, $\mathcal{P}$' output link for input link $l_i$ and destination $d$ is $l_{o_2}$. Then, routing according to multi-level communication consists in sending message $m$ over link $l_{o_2}$ of virtual network $vn_{i+1}$, and resuming routing of message $m$ in this virtual network according to $\mathcal{R}$ following the newly chosen path.

Thus, a $\mathcal{R}$ routing rule has been infringed but message $m$ has gone up by a communication level. Thus deadlock avoidance is conserved and no path elongation has occurred at processor $p$. This rule is applied at each processor, each times we *need* to keep minimal paths and as long as we have not reached the upper level.

## A.2.2 Analysis of the principle

It is obvious that the communication scheme as defined above improves the quality of the paths' stretch factor with regard to the primary scheme. Indeed, this new scheme allows lifting the restrictions induced by the communication on only one level. In this paragraph, we formulate this notion of communication levels and prove that deadlock avoidance is conserved for any set of $l$ ($l > 1$) levels.

### A.2.2.1 Definitions and notations:

Let:

- $G = (V, E)$ be the graph associated to the interconnection network:
  - $V$ is the vertices set, representing processors.
  - $E$ is the edges set, representing bidirectional links.
- $L = \{1, 2, \ldots, l\}$, be the set of communication levels ($l > 1$).
- $\Re = \{R_1, R_2, \ldots, R_r\}$, be the rules set defining the scheme $\mathcal{R}$.

For vertex $v$, we define routing in the network $G$ with the notion of communication levels, using a function in the following way:
**Definition 1:**

$$F_v : E \times L \times V \longrightarrow \wp(E \times L)$$
$$(l_i, n_c, d) \longmapsto (l_o, n_o)$$

$F_v$ gives, for input link $l_i$ of message $m$ in current level $n_c$ and to destination $d$, the set of all output links through which to send $m$ and the levels where it has to be stored. Routing on the graph $G$ is defined through function $F$:

**Definition 2:**

$$F = \bigcup_{v \in V} F_v$$

Let $\Gamma_v$ (resp. $P_v$) be the function corresponding to the scheme $\mathcal{P}$ (resp. $\mathcal{R}$):

**Definition 3:**

$$\begin{aligned} \Gamma_v : E \times V &\longrightarrow \wp(E) \\ (l_i, d) &\longmapsto l_o \end{aligned}$$

$$\begin{aligned} P_v : E \times V &\longrightarrow \wp(E) \\ (l_i, d) &\longmapsto l_o \end{aligned}$$

$\Gamma_v$ (resp. $P_v$) gives for input link $l_i$ and destination $d$, *all* output links $l_o$ leading to a minimal path (resp. computed according to the scheme $\mathcal{R}$). Function $F_v$ is then computed as follows:

- *if* $(P_v(l_i, d) \cap \Gamma_v(l_i, d)) \neq \emptyset$ *then* $F_v(l_i, n_c, d) = ((P_v(l_i, d) \cap \Gamma_v(l_i, d)), n_c)$
- *if* $(P_v(l_i, d) \cap \Gamma_v(l_i, d)) = \emptyset$ *then*
  *if* $n_c \neq l$ *then* $F_v(l_i, n_c, d) = (\Gamma_v(l_i, d), (n_c + 1))$
  *else* $F_v(l_i, n_c, d) = (P_v(l_i, d), n_c)$.

To prove deadlock avoidance, we need to make some remarks.

**Remark 1:**

The application $F_v$ can be split up into:

$$F_v = F_v^1 \cup F_v^{1 \succ 2} \cup F_v^2 \cup \cdots \cup F_v^{(l-1)} \cup F_v^{(l-1) \succ l} \cup F_v^l$$

such that:

$$(l_o, n_o) = F_v^i((l_i, n_c, d)) \implies n_c = n_o = i \quad \forall i, i = 1 \cdots l$$

resp,

$$(l_o, n_o) = F_v^{i \succ (i+1)}((l_e, n_c, d)) \implies n_c = i,\ n_o = i + 1 \quad \forall i, i = 1 \cdots (l-1)$$

i.e. the restriction of $F_v$ to level $i$ (resp. transitions from level $i$ to $i + 1$).

**Remark 2:**

Moreover, we have obviously that:

$$F_v^i \cap F_v^j = \emptyset \quad \forall i, j,\ \ i, j = 1 \cdots l$$

and,

$$F_v^{i \succ (i+1)} \cap F_v^j = \emptyset \quad \forall\, i = 1 \cdots (l-1),\ j = 1 \cdots l.$$

Next, we introduce the *dependency graph* associated to function $F$.

**Definition 4:**

A couple $(l_i, n_c)$ depends on $(l_o, n_o)$ through a precedence relationship that we note,
$(l_i, n_c) \prec (l_o, n_o)$, iff:

$$\exists\ v \in V,\ \ d \in V \quad / \quad F_v((l_i, n_c, d)) = (l_o, n_o),$$

Let $I_v$ be the set of incident edges to $v$, the dependency graph of function $F_v$ (resp. $F$) is defined as:

**Definition 5:**

$$D_{F_v} = \{\ (l_i, n_c) \prec (l_o, n_o)\ \ /\ \ l_i, l_o \in I_v\ ,\ n_c, n_o \in N\ \}$$

resp,

$$D_F = \bigcup_{v \in V} D_{F_v}$$

Then:

**Proposition:** *let, $I$ a processor interconnection network, $\Re$ a set of deadlock free routing rules on $I$ defining a primary communication scheme; then routing according to multi-level communication, on any set of $l$ levels ($l \geq 1$), associated with $\Re$, is deadlock free.*

### A.2.2.2 Proof of the proposition

To prove the proposition, we prove it for a 2-levels set. The generalization to a $k$-levels set can be deduced directly. First, we need to establish the following lemma:

**Lemma 1:** A cycle in $D_F$ corresponds to a *potential* deadlock state in the network $G$, and reciprocally.

**Proof ($\Leftarrow$):** suppose a deadlock situation between a set $\{v_j, j = 1 \cdots n\}$ of nodes. So, there is a *cycle of request*; each node contains a message $m_j$ and cannot send it to its neighbor $v_{(j+1)}$ (arithmetic operations are modulo $n$). Let $l_{i_j}$ (resp. $n_{c_j}$), be the link through which $m_j$ arrived to $v_j$ (resp. its current level). Let $l_{o_j}$ (resp. $n_{o_j}$), be the output link (resp. level) computed by $v_j$ to send message $m_j$ to its neighbor $v_{j+1}$. At node $v_j$ we have the following equality:

$$F_{v_j}((l_{i_j}, n_{c_j}, d_j)) = (l_{o_j}, n_{o_j})$$

$v_j$ can not send message $m_j$ to $v_{j+1}$ in three cases:

1. $n_{c_j} = 1, n_{o_j} = 1$ and $v_{(j+1)}$ can not receive message $m_j$ (its *virtual buffer* of level 1 corresponding to its *input* link $l_{o_j}$, is full),
2. $n_{c_j} = 1$, $n_{o_j} = 2$ and $v_{(j+1)}$ cannot receive message $m_j$ (its virtual buffer of level 2 corresponding to its input link $l_{o_j}$, is full),
3. $n_{c_j} = 2, n_{o_j} = 2$ and $v_{(j+1)}$ cannot receive message $m_j$ (for the same reason).

- assume the last case: at node $v_j$ the path $m_j$ has to take is such that:

$$F_{v_j}((l_{i_j}, 2, d_j)) = (l_{o_j}, 2),$$

likewise at node $v_{j+1}$, the path $m_{j+1}$ has to take is such that:

$$F_{v_{(j+1)}}((l_{o_j}, 2, d_{j+1})) = (l_{o_{(j+1)}}, 2)$$

and can not be sent to node $v_{j+2}$ (the virtual buffer of level 2 of this later is full). Thus, we have a succession of paths portions, described by the equation:

$$F_{v_j}((l_{i_j}, 2, d_j)) = (l_{o_j}, 2),$$

which loops on itself, because:

$$l_{o_j} = l_{i_{(j+1)}},$$

the output link of node $v_j$ is the input link of node $v_{j+1}$ and all virtual buffers of level 2 associated with entry links $l_{i_j}$ $j = 1 \cdots n$, are full. Therefore, at each node $v_j$ we have a dependency of the form:

$$(l_{i_j}, 2) \prec (l_{o_j}, 2)$$

this set of dependence closes up and forms a cycle in $D_F$ since:

$$l_{o_j} = l_{i_{j+1}},$$

- assume the second case: at node $v_j$, message $m_j$ must follow the path described by the equation:

$$F_{v_j}((l_{i_j}, 1, d_j)) = (l_{o_j}, 2),$$

i.e. message $m_j$ is stored in virtual buffer of level 1 of node $v_j$ and can not be sent to virtual buffer of level 2 of node $v_{j+1}$. At node $v_{j+1}$, message $m_{j+1}$

is stored in virtual buffer of level 2, so it can only be sent through a path such that:

$$F_{v_{j+1}}((l_{o_j}, 2, d_{j+1})) = (l_{o_{j+1}}, 2),$$

likewise for all messages $m_i$, $i > j+1$:

$$F_{v_i}((l_{i_i}, 2, d_i)) = (l_{o_i}, 2),$$

however, to have a request *cycle*, message $m_{j-1}$ of $v_{j-1}$ must be sent to the level 1 virtual buffer of node $v_j$. This implies the equation:

$$F_{v_{(j-1)}}(l_{i_{(j-1)}}, 2, d_{(j-1)})) = (l_{i_j}, 1)$$

which contradicts the computing scheme of function $F_p$. So, in this case we can not have a request cycle; i.e. a deadlocked situation.

- assume the first case: the argument is similar to case (3) and we have a deadlocked situation corresponding to a cycle in $D_F$.

**($\Rightarrow$) informally:** assume a cycle in $D_F$, so we have a series of dependencies which make a loop. Suppose that the virtual buffer corresponding to each pair $(l, n)$ of this loop contains a message, then we obviously reach a deadlocked situation. □

To prove the theorem, we have to prove that $D_F$ is cycle free.

**Lemma 2:** Dependancy graph $D_F$ is cycle free.

**proof:** for a 2-levels communication scheme:

$$F = \bigcup_{v \in V} F_p = \bigcup_{v \in V} (F_v^1 \cup F_v^{1 \succ 2} \cup F_v^2)$$

so,

$$D_F = \bigcup_{v \in V} D_{F_v} = \bigcup_{v \in V} (D_{F_v^1} \cup D_{F_v^{1 \succ 2}} \cup D_{F_v^2})$$

and,

$$D_F = (\bigcup_{v \in V} D_{F_v^1}) \bigcup (\bigcup_{v \in V} D_{F_v^{1 \succ 2}}) \bigcup (\bigcup_{v \in V} D_{F_v^2})$$

then,

$$D_F = D_{F^1} \cup D_{F^{1 \succ 2}} \cup D_{F^2}$$

$D_{F^1}$ is the dependency graph restricted to level 1, $D_{F^2}$ is the one to level 2 and $D_{F^{1 \succ 2}}$ is the one of level transitions (1 towards 2). $D_{F^1}$ and $D_{F^2}$ are dependency graphs computed according to the primary communication scheme because they are related to levels of communication. The primary communication scheme is

deadlock free so its graph dependency is cycle free (according to lemma 1) and therefore $D_{F_1}$ and $D_{F_2}$. $D_{F^{1\triangleright 2}}$ is defined by the equation:

$$D_{F^{1\triangleright 2}} = \{(l_i, 1) \prec (l_o, 2), \quad (l_i, l_o) \in E\}$$

we have seen previously that this type of dependencies cannot induce a cycle of request, since we can never close up a path that contains at least one dependency of this type. Then, we can never have a deadlock situation induced by this type of dependencies. Therefore, again according to lemma 1, $D_{F^{1\triangleright 2}}$ is cycle free. In summary, $D_{F_1}, D_{F_2}$ and $D_{F^{1\triangleright 2}}$ are cycle free, if taken separately. On the other hand, pairwise intersection between $F^1$, $F^2$ and $F^{1\triangleright 2}$ being empty (according to remark 2), it is the case for $D_{F^1}$, $D_{F^2}$ and $D_{F^{1\triangleright 2}}$. Then, there union does not induce a cycle. □

## A.3 Application

In this section, we are interested in applying multi-level communication based on *Eulerian cycle routing rules* [72] as a primary communication scheme to the $k$-ary $n$-cube with wrap around connections network.

The Eulerian cycle routing strategy computes an Eulerian cycle of the interconnection network[2] and routes according to *deadlock-free* rules [72] on this *routing structure.* This routing strategy induces *one* message storing buffer per outgoing link of a processor. That is, it uses *one communication level.* Thus, It is well adapted as a primary communication scheme for a multi-level communication strategy. Also, this strategy uses all network communication links. It guarantees, for some (*source*, *destination*) nodes pairs, routing through minimal paths. It also allows to envisage an adaptive routing. Meanwhile, in spite of its interesting routing properties, these are closely bounded to the quality of the Eulerian cycle used. In particular, the incidence of the Eulerian cycle on the paths' stretch factor is crucial. That is, the "*better*" is the Eulerian cycle, the closest to one will the paths' stretch factor be. Unfortunately, computing such a "*better*" cycle is prohibitive. Thus, beyond the efficient use of communication network resources, the multi-level communication is shown to be an interesting answer to the incidence problem of the Eulerian cycle on the paths' stretch factor. Particularly in the case of the $k$-ary $n$-cube with wrap around connections network[3].

The $k$-ary $n$-cube with wrap around connections network, namely torus[4] can be viewed as a $n$-dimensional mesh with $k$ nodes interconnected in a ring in

2. When the network admits one, otherwise, virtuals channels are used [72].
3. This network admits an Eulerian cycle since all its nodes are of even degree $2n$ [8].
4. In what follows, we will use the notation $torus(n, k)$ to denote such a network.

each dimension. More formally, this topology is defined as the *cartesian product* of $n$ $k$-nodes rings[5]. This network has been used in several significant parallel computers like the COSMIC CUBE [23], the CONNECTION MACHINE [54] and more recently the T3D [18]. It has interesting topological properties, and has proven to be useful for general purpose processing. High radix ($k$), low dimensional ($n$) networks are especially easy to construct and scale [22]. The most used routing algorithm for this network is *e-cube* routing. Unfortunately this algorithm is not deadlock free. To devise deadlock free algorithms, several investigators have used virtual channels [23]. The resulting algorithm either does not ensure routing on shortest paths or needs a large amount of memory space for representing the routing function or storing users messages. In [82], SAKHO&*al* reported an interval labeling scheme which allows deadlock free routing with constant memory space. But it does not ensure shortest paths routing. Another approach for deadlock avoidance is given by GLASS and NI in [44]. Their model is not based on the concept of virtual channels, instead it consists in analyzing the direction at which packets can turn and the cycles that the turns can induce and then by prohibiting just enough turns to break all possible cycles. Their algorithm does not ensure shortest paths routing too. A shortest paths, deadlock free and fully adaptive algorithm is given in [69] by LINDER and HARDEN. This algorithm needs $(n+1)2^{n-2}$ virtual channels per physical one. Thus, it requires a very large memory space for each processor. CYPHER and GRAVANO state in [20] that *fully adaptive*, minimal, deadlock free routing for this network can be carried out with just 2 buffer queues per incoming link and gave an algorithm ensuring such properties. In contrast, their algorithm can require, at each node, two routing tables of $O(k^n)$ memory space for representing the routing function in one hand, and is such that its channel dependency graph is not cycle free. Thus, this algorithm is not based on a strong condition to prevent deadlocks. This condition is in fact necessary as we have seen it in the introduction. In this section, we prove that Eulerian cycle routing based multi-level communication leads to a *deterministic, minimal, deadlock free* routing algorithm which requires 2 *communication levels* (2 virtual channels per physical one) using a *constant* memory space for representing the global routing strategy. We first reminder the Eulerian cycle routing strategy.

### A.3.1 The Eulerian cycle routing strategy

Let $G = (V, E)$ be the graph associated with the interconnection network.
**Definition 1 :**

5. In what follows, we will denote such ring by $k$-cycle.

Routing on graph $G$ is defined by a function $R$ such as:

$$R : E \times V \times V \longrightarrow E$$
$$(l_i, s, d) \longmapsto l_o$$

where, $l_i$ is the input link of a message $m$ into node $s$ and $l_o$ the output link computed by $s$ to deliver $m$ to destination $d$. Then, we say that $l_i$ *depends* on $l_o$.

**Definition 2:**
the dependency graph of function $R$ is defined as:

$$D = (E, E'),$$

where $E'$ is the set of depending link pairs of $E$.

According to DALLY and SEITZ theorem [23], routing function $R$ on graph $G$ is deadlock free if $D$ is cycles free. This is the idea behind the Eulerian cycle routing strategy for deadlock avoidance [72]. Now, we introduce the routing strategy. Let $G' = (V', E')$ be an Eulerian traversal associated with graph $G$. $G'$ is a closed traversal, started from a root node $r$, which visits all $G$'s edges exactly one time [8]. Let $f$ be a numbering of $G'$'s vertices defined as follows:

$$f : V' \longrightarrow N \times N$$
$$v \longmapsto (f_1(v), f_2(v))$$

where:

- $f_1(r) = 0$
- $f_1(v)$ represents the *ordinal* of $v$ in the Eulerian tour.
- $f_2(v)$ represents the processor number of occurrence $v$.

We define the followings:

- each *edge* of $G'$ is labelled by the number of its *initial* vertex according to an $f_1$ increasing $G'$'s vertices traversal.
- $G'_+$ (resp. $G'_-$) as the *directed graph* obtained from $G'$ when trailing its vertex in an $f_1$ increasing (resp. decreasing) order.
- an *arc*[6] of $G'_+$ (resp. $G'_-$) is said to be *direct* (resp. indirect).
- each *arc* of $G'_+$ (resp. $G'_-$) is labeled by the number of its $G'$s corresponding *edge*.

6. An arc is an oriented edge.

Assuming that to *each arc* is assoiated a message storing buffer at its final extrimity, the routing strategy consists to respect the following dependencies rules:
**Dependencies Rules:**

- a direct arc can not depend on an indirect one.
- a direct arc can not depend on a less numbered direct one.
- an indirect arc can not depend on a great numbered indirect one.

Thus, function $R$ can only induce paths of the following three types:

- an increasing sequence of direct arcs.
- a decreasing sequence of indirect arcs.
- a decreasing sequence of indirect arcs followed by an increasing sequence of direct arcs.

#### A.3.1.1 An useful improvement of Eulerian cycle routing rules

Let define the following improvements of the routing strategy described above:

1. consider the same definitions as above, with the modifications:
2. $f_1(r) = 1$.
3. each *arc* of $G'_+$ (resp. $G'_-$) is numbered by $+$(the number of its corresponding $G'$'s *edge*), (resp. $^-$(the number of its $G''$ *edge*)).

The routing strategy is then modified on the following *single* rule:
**Dependency Rule:**

- any arc can not depend on a *less numbered* arc.

and the function $R$ can only induce paths of the following form:

- *an increasing sequence of arcs.*

Thereby, the Eulerian routing strategy consists, in computing an Eulerian cycle of the graph $G$, to label the direct and indirect arcs according to the modification described above and to route according to an increasing sequence of arcs. In the next paragraph we will exhibit a particular Eulerian traversal of the torus and prove that, the application of Eulerian cycle routing based multi-level communication on this cycle, with the above modifications, leads to an optimal storage, deterministic routing algorithm for such networks.

### A.3.2 Eulerian cycle routing rules based Multi-level communication for torus networks

In an orthogonal axis system of cardinality $n$, each node $x$ of a $torus(n,k)$ may be defined by its $n$ coordinates $(x_0, x_1, ..., x_{n-1})$ belonging to the set $E_k = \{0, 1, 2, ..., k-1\}$. Each node is connected to every other one whose $n$ coordinates differ in exactly one position by $(\pm\ 1\ mod\ k)$, and is in the intersection of $n$ $k$-cycles; one $k$-cycle per dimension. According to this representation, let the Eulerian traversal defined by the following *recursive* procedure:

```
Procedure Torus_Euler_Cycle(n,k,(0,...,(n-1)))
Begin
  index=0;
  While (index<(k-1))
    Do
      If (n>1) Then
                 Goto from node (0,...,0,index) to node
                                          (0,...,0,index+1);
                 Torus_Euler_Cycle((n-1),k,(0,...,(n-2)));
               Else
                 loop the k-cycle of dimension 0 according
                 to an increasing order of the coordinate;
                 index=k+1;
      Endif
      index=index+1;
    Enddo
    If (n>1) Then
               Goto from node (0,...,0,(k-1)) to node (0,...,0);
               Torus_Euler_Cycle*((n-1),k,(0,...,(n-2)));
    Endif
End
```

**Comment:** This procedure performs an Eulerian traversal of a $torus(n,k)$ ($n$ and $k$, being the first input parameters of the procedure) represented in an axis system whose dimensions are $0, ..., (n-1)$ (the last input parameter of the procedure) (see figure A.3). Starting the traversal from root node of coordinate $(0, ..., x_{n-1} = 0)$, we pass to node of coordinate $(0, ., 0, x_{n-1} = 1)$ in the last dimension $n-1$. From this later node, we perform *recursively* an Eulerian traversal of the $torus(n-1,k)$ whose all nodes have the coordinate $x_{n-1} = 1$. This traversal will terminates at node $(0, ., 0, x_{n-1} = 1)$. Then, we pass to node $(0, ., 0, x_{n-1} = 2)$ in the dimension $n-1$. As in the previous case, from this later node, we perform recursively

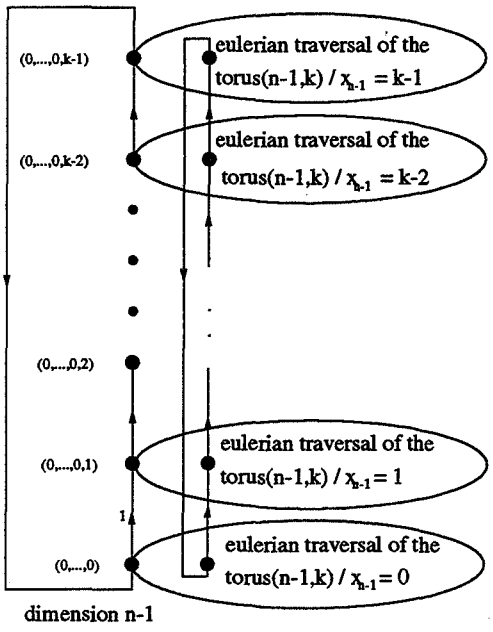


FIG. A.3 – *An Eulerian traversal of the torus*$(n, k)$.

an Eulerian traversal of the *torus*$(n-1, k)$ whose all nodes have the coordinate $x_{n-1} = 2$. And so on. While, the Eulerian traversal of the *torus*$(n-1, k)$ whose all nodes have the coordinate $x_{n-1} = k-1$ have been performed, we pass from node $(0, ., 0, x_{n-1} = k-1)$ to node $(0, ., 0, x_{n-1} = 0)$ on the wrap around connection, in the dimension $n-1$. Then, we perform recursively again, an Eulerian traversal of the *torus*$(n-1, k)$ whose all nodes have the coordinate $x_{n-1} = 0$, with the hypothesis (which corresponds to $(*)$ in the recursive procedure call): at each node $x$ of this *torus*$(n-1, k)$, the $k$ – *cycle* of dimension $n-1$ is *looped when traversing the node x for the last time*[7], following an increasing order traversal of the $x_{n-1}$ coordinate. □

Figure A.4 shows an example of a *torus*$(n, k)$ with $n = 2$ and $k = 5$. It allows to see the Eulerian traversal with the numbering of the edges[8]. This particular Eulerian traversal allows the following result:

7. That is, just after the dimension $n-1$ $k$-cycle traversal, the node $x$ will be leaved for the last time. This order traversal of the dimension $n-1$ $k$-cycles at the nodes whose $x_{n-1} = 0$, is *crucial* for the proof of the theorem given in this section.

8. Which corresponds to the numbering of the direct arcs, indirect arcs are numbered with the opposite number.

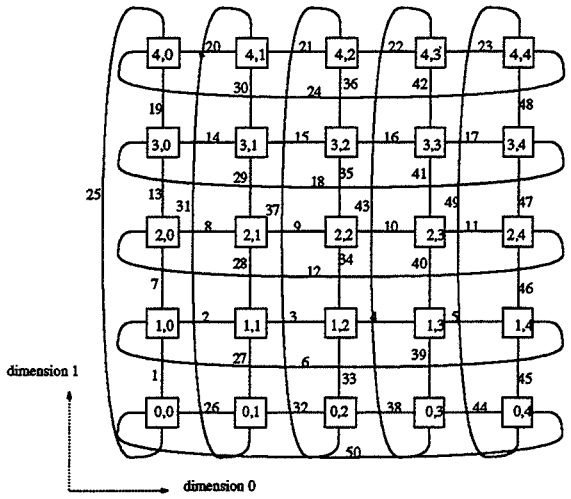


FIG. A.4 – *A 5-ary 2-cube with wraparound connections topology.*

**Theorem:** *Let $C$ be the Eulerian traversal, according to the procedure described above, of a torus$(n,k)$ for $n \geq 1$, $k \geq 1$. Routing according to multi-level communication based on Eulerian cycle routing rules using cycle $C$, routes on minimal paths with 2 communication levels.*

### A.3.2.1 Proof of the theorem

Let $C^+$ be the oriented graph defined by the procedure direction traversal. $C^-$ is the oriented graph traversed in the reverse direction. Nodes and edges of the torus are numbered according to function $f_1$ during the direct traversal (see figure A.4 for the numbering of the direct arcs). Let define the following notations:

- $C_k$, a $k$-cycle.
- $C_{ki}$, a $k$-cycle of dimension $i$, $\forall\ i = 0...n-1$.
- $C_{ki}^+$ (resp. $C_{ki}^-$), a $k$-cycle of dimension $i$ oriented in the direct (resp. indirect) direction.

Figure A.5 shows the traversal of a $C_{ki}$. It allows to see all the dependencies allowed while routing on a $C_{ki}$. The figure shows the direct orientation[9], namely $C_{ki}^+$. In particular, routing on $C_{ki}^+$ (resp. $C_{ki}^-$) according to an $f_1$ increasing order,

9. In fact, the traversal of a $C_{ki}^+$ in the general case; the boundary cases respect *exactly* the same dependencies forms.

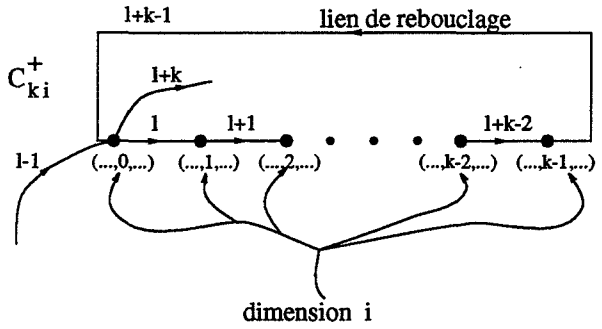


FIG. A.5 – *An $f_1$ labeling of a $C_{ki}$.*

is allowed. Note also, that the dependency between the wrap around connection (labeled $l+k-1$) and the first link (labeled $l$), in node of coordinate $(., x_{i-1} = 0, .)$ is not allowed in the two directions; according to Eulerian cycle routing rules.

It is obvious (see [52]) that the length of a minimal path, in a $torus(n,k)$, from source node $s = (s_0, ..., s_{n-1})$ to a destination node $d = (d_0, ..., d_{n-1})$ is given by:

$$\sum_{i=0}^{n-1} Min(|C^+_{ki}(s_i, d_i)|, |C^-_{ki}(s_i, d_i)|)$$

where, $|C^+_{ki}(s_i, d_i)|$ (resp. $|C^-_{ki}(s_i, d_i)|$) is the length of a path on $C_{ki}$ from source node of coordinates $(s_0, ..., s_{n-1})$ to destination node of coordinates $(s_0, ., d_i, ., s_{n-1})$ according to the direct (resp. indirect) direction. To prove the theorem, we need the following lemmas:

**Lemma** 1 : *Routing on a $torus(n,k)$ for $n \geq 1$, $k \geq 1$, according to multi-level communication based on Eulerian cycle rules using cycle $C$, routes on minimal paths from any source node $s$ to destination node $(0, ..., 0)$ with* **one** *communication level.*

**Proof:** by induction on $n$:

- n=1 : In this case the $torus(n,k)$ is reduced to a $k$-cycle. On this cycle, at each node, all the dependencies are allowed except at the root node, where they are prohibited in the two directions. It is obvious that minimal routing is always possible from any source node $s$ to node $(0)$ with one communication level since this latter node is the *destination.*

- suppose the lemma is true for a $torus(n-1,k)$: Let $m$ be a message to route from source node $s=(s_0,...,s_{n-1})$ to destination node $d=(0,...,0)$ on a minimal path of a $torus(n,k)$. It is obvious that minimal routing in a $torus(n,k)$ from $s$ to $d$ corresponds to minimal routing in the $torus(n-1,k)$ defined by all the nodes $x=(x_0,...,x_{n-1})$ such that $x_{n-1}=s_{n-1}$, from node of coordinates $(s_0,...,s_{n-2})$ to node $(0,...,x_{n-2}=0)$ and minimal routing on the $C_{k(n-1)}$ containing node $d$. We distinguish two cases:

  1. minimal routing on $C_{k(n-1)}$ is according to $C^+_{k(n-1)}$: We route from s to node $i=(0,...,x_{n-2}=0)$ in the $torus(n-1,k)$ defined by $x_{n-1}=s_{n-1}$, using one communication level (the first) (induction hypothesis), then from $i$ to $d=(0,...,x_{n-1}=0)$ in the $torus(n,k)$, according to the $C^+_{k(n-1)}$. Note that at node $i$, there is no level transition, since the input arc of message $m$ is always less numbered than its output arc.
  2. minimal routing on $C_{k(n-1)}$ is according to $C^-_{k(n-1)}$: We route from s to node $i=(s_0,...,s_{n-2},0)$ in the $torus(n,k)$, according to the $C^-_{k(n-1)}$, then from $i$ to $d=(0,...,x_{n-2}=0)$ in the torus $torus(n-1,k)$ defined by $x_{n-1}=0$ using one communication level (induction hypothesis). Note that at node $i$, there is no level transition, since the input arc of message $m$ is less numbered than its output arc. □

**Lemma 2**: *Routing on a $torus(n,k)$ for $n\geq 1$, $k\geq 1$, according to multi-level communication based on Eulerian cycle rules using cycle $C$, routes on minimal paths from source node $(0,...,x_{n-1}=0)$ to any destination node $d$ with* **one** *communication level.*

**Proof:** It is based on the proof of the previous lemma. As Eulerian cycle routing rules is a symmetrical routing strategy [72]; that is, if $\mu$ is a correct routing path from $s$ to $d$, then the same routing path (in the reverse direction) is a correct one from $d$ to $s$. Then, to route from source node $s=(0,...,0)$ to any destination node $d$ using one communication level, it just suffices to use the routing path from $d$ to $s$ in the reverse direction. □

Again for the theorem, we make a proof by induction on $n$ and prove that at most *one level transition* is sufficient to have minimal routing paths:

- $n=1$: In this case the $torus(n,k)$ is reduced to a $k$-cycle. On this $k$-cycle, at each node all the dependencies are allowed except at root node of coordinate 0, where they are prohibited in the two directions. It is well known [52, 20] that with two virtual channels per physical one (two communication levels) minimal routing is possible.

- suppose the theorem is true for a $torus(n-1)$: Let $m$ be a message to route from source node $s = (s_0, ..., s_{n-1})$ to destination node $d = (d_0, ..., d_{n-1})$ on a minimal path of a $torus(n,k)$. It is obvious that minimal routing from $s$ to $d$ on a $torus(n,k)$ corresponds to minimal routing on a $torus(n-1,k)$ and minimal routing on a $C_{k(n-1)}$ of the last dimension. Again, We distinguish two cases:

  1. minimal routing in last dimension $n-1$ is according to $C^+_{k(n-1)}$: Consider minimal routing, in the $torus(n-1,k)$ defined by $x_{n-1} = s_{n-1}$, from source node of coordinates $(s_0, ..., s_{n-2})$ to destination node of coordinates $d_1 = (d_0, ..., d_{n-2})$ (see figure A.6). Assume the worst case, this routing needs two communication levels. Let $i = (i_0, ..., i_{n-2})$ be the node where the level transition take place. Let $j = (i_0, ..., i_{n-2}, 0)$ and $d_2 = (d_0, ..., d_{n-2}, 0)$ be nodes of the $torus(n-1,k)$ defined by $x_{n-1} = 0$. Then routing is performed according to the following path: we route from $s$ to $i$ following the minimal $(s, d_1)$ routing path. Then from $i$ to $j$ in the $torus(n,k)$, following a $C^+_{k(n-1)}$. Note that at node $i$, *there is no level transition, since the input arc traversed by m (the incident arc to i) is less numbered than the output arc traversed after i.* Then from $j$ to $d_2$, making a level transition at $j$, in the $torus(n-1,k)$ defined by $x_{n-1} = 0$. This routing path is exactly similar to the routing path in the $torus(n-1,k)$ defined by $x_{n-1} = s_{n-1}$ from $i$ to $d_1$. Finally, from $d_2$ to $d$, in the $torus(n,k)$. Note that at $d_2$ no level transition is necessary, except if $d$ belongs to the $C^+_{k(n-1)}$ defined by $(0,.,0,x_{n-1} = 0...(k-1))$. This implies that $d$ is of coordinates $(0, ..., 0, d_{n-1})$. In this case, we use the previous lemmal and route from $s$ to the root node $d_1 = (0,.,x_{n-2} = 0)$ of the $torus(n-1,k)$ defined by $x_{n-1} = s_{n-1}$ using one communication level. Then from $d_1$ to $d$ on the $C^+_{k(n-1)}$ defined by $(0,.,0,x_{n-1} = 0...(k-1))$ making a level transition at node $d_2$. Again, note that there is no level transition at node $d_1$.

  2. minimal routing in last dimension $n-1$ is according to $C^-_{k(n-1)}$: Consider minimal routing in the $torus(n-1,k)$ defined by $x_{n-1} = s_{n-1}$, from source node of coordinates $(s_0, ..., s_{n-2})$ to destination node $d_1 = (d_0, ..., d_{n-2})$ (see figure A.7). Assume the worst case, this routing needs two communication levels. Let $i = (i_0, ..., i_{n-2})$ be the node where the level transition take place. Let $i_1$, $i_2$ and $s_1$ be the nodes of coordinates $i_1 = (i_0, ..., i_{n-2}, 0)$, $i_2 = (i_0, ..., i_{n-2}, d_{n-1})$ and $s_1 = (s_0, ..., s_{n-2}, 0)$. Then routing is performed according to the following minimal path: we route from $s$ to $s_1$ in the $torus(n,k)$ following a $C^-_{k(n-1)}$. Then from $s_1$ to $i_1$ in the $torus(n-1,k)$ defined by $x_{n-1} = 0$. This routing path

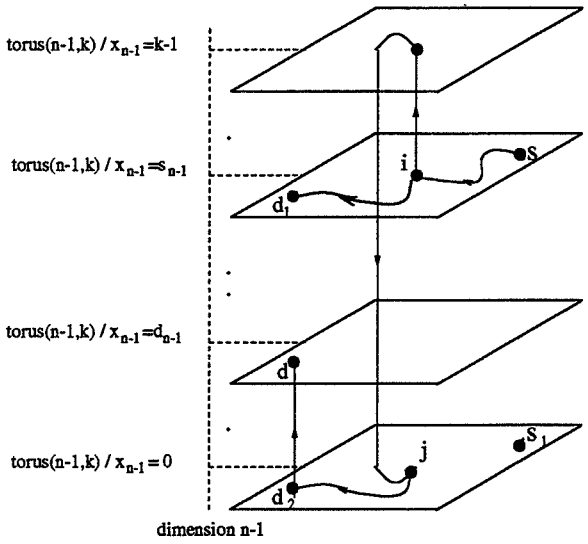


FIG. A.6 – *Routing in dimension n-1 is according to* $C^{+}_{k(n-1)}$.

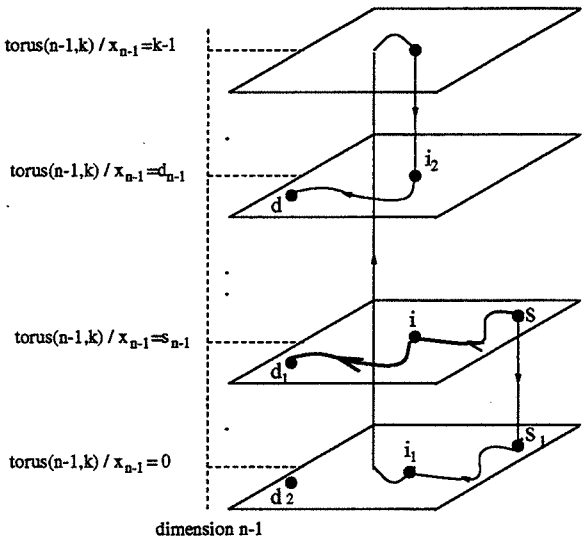


FIG. A.7 – *Routing in dimension n-1 is according to* $C^{-}_{k(n-1)}$.

is exactly similar to the routing path in the $torus(n-1,k)$ defined by $x_{n-1} = s_{n-1}$ from $s$ to $i$. Note that at node $s_1$ there is no level transition since the input arc traversed by $m$ (the incident arc to $s_1$) is less numbered than the output arc traversed after $s_1$; except if $s$ belongs to the $C^-_{k(n-1)}$ defined by $(0,.,0,x_{n-1} = 0...(k-1))$, we treat this case at the end of this paragraph. Then we route from $i_1$ to $i_2$ in the $torus(n,k)$, following a $C^-_{k(n-1)}$, making a level transition at node $i_1$. Finally from $i_2$ to $d$ in the $torus(n-1,k)$ defined by $x_{n-1} = d_{n-1}$. This routing path is exactly similar to the routing path in the $torus(n-1,k)$ defined by $x_{n-1} = s_{n-1}$ from $i$ to $d_1$. Note that at node $i_2$ there is no level transition.

If $s$ is of coordinates $(0,...,0,s_{n-1})$, we use lemma2 and route from $s$ to root node $i_2 = (0,.,x_{n-2} = 0)$ of the $torus(n-1,k)$ defined by $x_{n-1} = d_{n-1}$, on the $C^-_{k(n-1)}$ defined by $(0,.,0,x_{n-1} = 0...(k-1))$, making a level transition at node $s_1$. Then from $i_2$ to $d$ using one communication level (lemma2). Again, note that there is no level transition at node $i_2$. □

This completes the proof of the theorem. This theorem allows to infer a *deadlock-free, deterministic, minimal, storage optimal* routing algorithm for the $torus(n,k)$. In fact, given a message $m$ to send from $s$ to $d$, it is always routed following an *increasing sequence of arcs* according to Eulerian cycle routing rules. At an intermediate node $i$, given the label $l$ of the input arc, message $m$ is sent through an output arc leading to minimal routing whose label is greater than $l$. If such arc does not exist, then a level transition is carried out. Now, it remains two answer the following two questions:

1. if there are several allowed output arcs[10] leading to minimal routing, which one must be chosen?

2. when making a level transition, if there are several output arc leading to minimal routing, which one must be chosen?

Concerning the first question, it is obvious that the right arc is the one of the smallest label. Namely, if $\{l_i, i = 1...m\}$ is the set of the allowed arc labels, the one which must be chosen is defined by: $\underset{i=1...m}{\text{Min}} \{l_i\}$. In fact, this choice is *necessary* to have at most one level transition (when required) in the routing path between $s$ and $d$. That is, to avoid a supplementary level transition because of a false choice, and then to be forced to use more than two communication levels in a $(s,d)$ routing path. For the second question, the right arc is again the smallest

10. Recall that an allowed output arc is one whose label is greater than $l$.

label one, for the same reason. In fact, this problem arises at the source node $s$; where, a choice is to be made with regard to the first dimension (if many) in which to send message $m$. The smallest label arc has to be chosen.

### A.3.2.2 A deterministic storage optimal routing algorithm

We suppose that during the Eulerian traversal, all edges of the $torus(n,k)$ are correctly numbered. Each node keeps the number of all its adjacent arcs; that is the direct and indirect ones.

Let us now describe the algorithm that a node $s = (s_0, ..., s_{n-1})$ has to execute to forward a message. Let *Labels* be the set of $s$' output arcs[11] labels . The algorithm supposes also that a message *msg* consists of a destination node $d = (d_0, ..., d_{n-1})$, a data to be transmitted and the level in which the message must be stored at the next node. Let $l$ be the label of the input arc of message *msg*.

```
begin
   /* compute the dimensions in which msg could be forwarded */
   Dim = {i ∈ E_k, s_i ≠ d_i};
   /* eliminate the label of the opposite arc to the msg input one */
   L = Labels − {the label of the opposite arc to arc labeled l};
   /* eliminate all arcs that are not according to a dimension in Dim */
   L = L − {all labels of arcs not according to a dimension in Dim};
   /* eliminate all arcs which does not lead to a minimal routing
              path with regard to the destination d */
   L = L − {all labels of arcs not leading to minimal routing};
   /* compute the label of the output arc on which to send message msg */
   lab = Min_{l_i∈L} l_i;
   /* decide if a level transition is necessary and send message msg */
   if (s is not the source of message msg) then
      if (l < lab) then
         send msg on arc of label lab on the same level;
      else
         send msg on arc of label lab on the next upper level;
   else
      send msg on arc of label lab on the first level;
end
```

11. Arcs such that node $s$ is there initial vertex.

## A.4 Discussion

In this paper, we have presented a general methodology for the improvement of the qualities of a given communication scheme. This methodology conserves the primary communication scheme qualities, particularly correction ones.

This approach is fairly generic to envisage a correlation between other *(correction, efficiency)* pair criteria. It is for example the case of (*deadlock avoidance*, *adaptive routing*) or (*deadlock avoidance*, *physical fault tolerance*). Indeed, the idea of generacity is according to the following optic: a communication level ensures a correction criteria, the replication-ordering of these levels will allow to integrate an efficiency criteria.

When the pair is (*deadlock avoidance*, *paths' stretch factor*), the multi-level communication allows to reduce the communication delay of the routing paths. In addition, the methodology allows to measure the space memory required to have a unitary stretch factor. Meanwhile note that in this case, the methodology is not well adapted if the primary communication scheme prevents deadlocks by constructing directed acyclic buffers graphs[12]. Indeed, for such primary schemes the memory size required at each processor to avoid deadlock is generally depending on the network size. Thus, the replication of this scheme will increase this size with the number of communication levels; so resulting on a large memory space size.

When the primary communication scheme is routing by Eulerian cycle rules, the approach allows to reduce the incidence on the paths' stretch factor of the *hard* problem of the "best" Eulerian cycle. The application of the resulting communication scheme to the torus network has validated the global methodology. In fact, the theorem of section 3 means that deadlock-free, minimal routing on a $torus(n, k)$ can be carried out with a number of level independent from the network size. That is, a memory space for storing users messages independent from the network size. Moreover, this space is optimal, it equals 2. As it has been proved [20, 52] that minimal, deadlock-free routing on a cycle requires *at least* two virtual channels per physical one, i.e. two communication levels, this is necessarily the case for the torus network[13]. More especially as our constructing methodology is based on a strong condition to prevent deadlocks; i.e. an acyclic dependency graph.

From the theorem, we have deduced a deterministic, storage *optimal* routing algorithm for the torus network. In fact, the memory space necessary for representing the routing strategy is constant. It just consists to keep in each node the

12. Assuming that the primary scheme do not ensure a unitary stretch factor.
13. Since it is the cartesian product of cycles.

labels of all its incident direct and indirect arcs. This is a very small memory space. In other hand, the memory space for storing users messages being optimal; thus the global memory space is optimal. We do not know such a routing algorithm in the literature. Meanwhile, this algorithm does not ensure adaptive routing; i.e. it does not allow several routing paths, given a (*source*, *destination*) nodes.

To ensure this latter routing criteria, we use more communication levels with the same philosophy. That is, we go up by a communication level each time we would like to ensure adaptivity. Thereby, with greater than 2 communication levels, the result of the methodology is to allow as adaptive deadlock-free, minimal routing as desirable. In this case, the routing strategy is represented, at each node, by one routing table whose size equals $O(k^n)$. This routing table is computed using an *inductive* algorithm whose induction bears on the length of the routing paths [72], and which takes into account the transition of levels. Then we obtain, a deadlock-free, minimal, adaptive routing algorithm with a network size independent (but greater than 2 levels) memory space for storing users messages and *one* routing table of $O(k^n)$ for representing the routing strategy.

# Annexe B

# Minimal, Deadlock Free and $O(n)$ Space Memory Routing for $k$-ary $n$-cubes with Wraparound Connections [52]

B.HADIM I.SAKHO
Centre SIMADE - Ecole des Mines de St Etienne
158 Cours Fauriel 42023 St Etienne Cedex 2 France
e-mail: {sakho,hadim}@emse.fr - tel: 04 77 42 01 72 - fax: 04 77 42 66 66

Abstract

This paper deals with implicit deadlock free routing on shortest paths for networks of processors interconnected in $k$-ary $n$-cube with wraparound connections topology. It is proved that such routing can be carried out with at least $(n+1)$ queues per outgoing communication link. The proposed algorithm is an extended version, for the torus network, of the $e$-cube routing algorithm. As in practice $n$ does not exceed 3, the algorithm is well-suited for VLSI implementation. To ensure deadlock free routing on *all* the shortest paths, less than $(n\lfloor k/2 \rfloor - 1)$ queues per outgoing link are required so allowing to envisage adaptive and fault tolerant routing.[1]

*Keywords:* Regular interconnection network, Deadlock free routing, Shortest paths routing, Adaptive routing and Fault tolerant systems.

1. Ce papier a été publié dans les actes de la conférence internationale: *Parallel and Distributed Processing: Techniques and Applications.* 3-4 Novembre 1995, Athens-Georgia USA. Il est également disponible en rapport interne RR 95.12 du centre SIMADE-ENSMSE.

## B.1 Introduction

Interprocessor communication is one of the key issues for the efficient use of multiprocessor computers. In Distributed Memory MIMD (DM-MIMD) architectures, processors are interconnected by a set of point to point communication links according to a certain topology and communicate by exchanging messages through these links. In this case communications between distant processors need careful attention. This has been responsible for the development of interconnection networks, routing algorithms and hardware routers [54, 18, 22, 23] that facilitate such communications.

When devising a routing algorithm, there are several criteria which must be considered. We can cite for instance, *deadlock avoidance, livelock, fault tolerance*, routing on *shortest paths* and minimal memory space for storing *users messages* and *routing information.* Deadlock avoidance is the most important criterion. Deadlock is a situation where some messages are *waiting* for an event which can not happen. This is for instance the case when a set of processors develops a *dependency loop* among themselves that prevents further messages movement; each processor tries to send its message to its neighbor in the loop.

An other important criterion is the length of the paths between any couple of (*source*, *destination*) processors which should be minimal. Indeed, the length of the paths influences directly the communication delay. So, the shorter they are, the shorter will be the communication delay.

Deadlocks can be prevented by restricting the use of communication links. This means that at a given node, some outgoing links can not be the successors of an incoming link in the routing paths. Such restrictions can result in stretching the length of these routing paths in regard to the shortest ones. Virtual channels introduced by DALLY and SEITZ [24, 23] is the most used concept for deadlock avoidance.

On other hand, minimal routing, that is routing according to the shortest paths of the interconnection network, supposes no restrictions between incoming and outgoing links at any processor. This lack of restriction can result on dependency loops among some subsets of processors. In consequence these two criteria are conflicting. Routing on shortest paths while avoiding deadlock implies for every processor a large number of queues, causing a large buffer space for storing messages.

The $k$-ary $n$-cube with wraparound connections can be viewed as a $n$ dimensional mesh with $k$ nodes interconnected in a ring in each dimension. More formally, this topology is defined as a *cartesian product* of $n$ $k$-nodes rings. Each node $x$ of such a network may be defined, in an orthogonal axis system of cardinality $n$, by its $n$ coordinates belonging to the set $E_k = \{0, 1, 2, ..., k-1\}$. Each node is

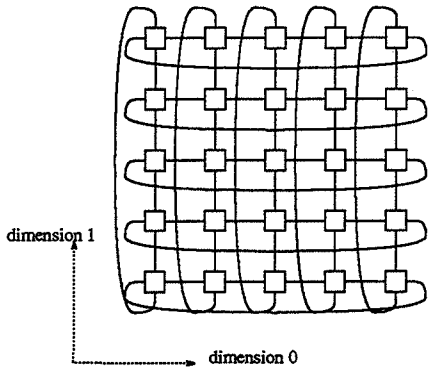


FIG. B.1 – *A 5-ary 2-cube with wraparound connections topology.*

connected to every other one whose $n$ coordinates differ in exactly one position by ($\pm$ 1 *mod* $k$). Figure 1 shows an example of such a network with $n = 2$ and $k = 5$.

This kind of interconnection network has been used in several significant computers like the COSMIC CUBE [23], the CONNECTION MACHINE [54] and the T3D [18]. The most used routing algorithm for this network is the *e-cube* algorithm. Unfortunately this algorithm is not deadlock free. For devising deadlock free algorithms, several investigators have used virtual channels [23]. The resulting algorithm either does not ensure routing on shortest paths or needs a large amount of space memory for representing the routing function or for storing the users messages.

In [82], SAKHO*&al* reported an interval labeling scheme for this network that allows deadlock free routing with constant memory space. But it does not ensure shortest paths routing. An another approach for deadlock avoidance in this network is given by GLASS and NI in [44]. Their model is not based on the concept of virtual channels, instead it consists in analyzing the direction in which packets can turn in the network and the cycles that the turns can induce and then by prohibiting just enough turns to break all of the possible cycles. But, their algorithm also does not ensure shortest paths routing. A shortest paths, deadlock free and fully adaptive routing algorithm is given in [69] by LINDER and HARDEN. In return, this algorithm needs $(n+1)2^{n-2}$ virtual channels per physical one. So, it requires a very large memory space at each processor. CYPHER and GRAVANO state in [20] that minimal, deadlock free routing for this network can be carried out with just 2 buffer queues per incoming link and give an algorithm ensuring such properties. In contrast, their constructing methodology requires $O(k^n)$ space memory for representing the routing function in one hand, and needs to transfer

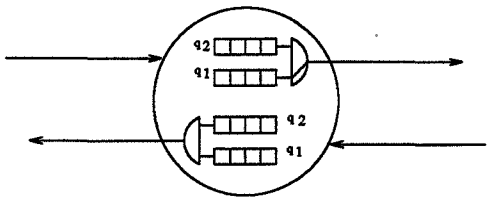


FIG. B.2 – *A node of a k-nodes ring: $q_1$ is the queue associated to each outgoing link.*

messages between internal queues of a node for routing on shortest paths in other hand. So the communication delay between a pair of (*source*, *destination*) nodes on the shortest path is extended.

The purpose of this paper is to present for the $k$-ary $n$-cube with wraparound connections an implicit routing algorithm, that is, constant memory space for representing the routing function, which is deadlock free and routes on the shortest paths while requiring $(n+1)$ virtual channels per physical one.

Section 2 presents the fundamentals of the routing algorithm. Deadlock avoidance for the simplest $k$-ary $n$-cube, the $k$-nodes ring is studied. It is proved that when $k > 4$, two queues (one being the queue by default) are needed to ensure deadlock free routing on all the shortest paths . In section 3, this study is extended to the $k$-ary $n$-cube network with $n \geq 2$ as a cartesian product of $n$ $k$-nodes ring. The main result of this extension is that at least $(n+1)$ queues per outgoing channel suffice for deadlock free routing on shortest paths. Taking into account other properties such as deadlock free routing on *all* the shortest paths require more than $(n+1)$ queues per outgoing link. In section 4 it is proved that less than $(n\lfloor k/2 \rfloor - 1)$ queues per outgoing link would suffice. That allows, to envisage adaptive even fault tolerant routing on shortest paths.

## B.2 Fundamentals of the algorithm

To understand the fundamentals of the algorithm, consider the simplest form of the $k$-ary $n$-cube obtained for $n = 1$. In this case the network is reduced to a $k$-nodes ring, where each node $x$ is defined by its unique coordinates $x_i$. Let the increasing (decreasing) order of the coordinates, be the positive (negative) direction of the ring; we will note $dir = 1$ for the positive direction and $dir = 0$ for the negative one. Associate to each outgoing link of node $x$ one queue $q_1$ for storing the messages which must be transmitted on that link (see Figure 2). Now suppose that each node $x_i$ contains a message, whose size equals the size of the

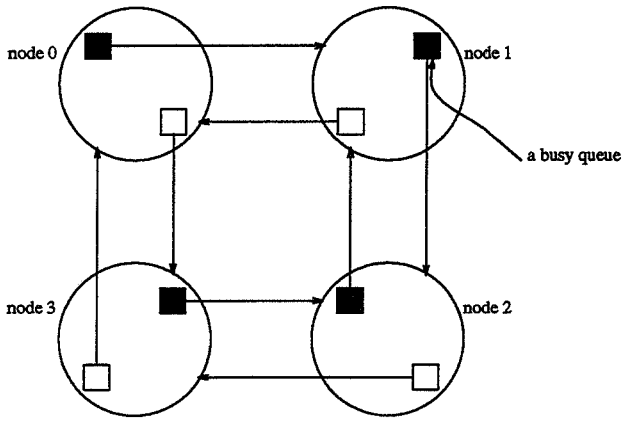


FIG. B.3 – *Each node i has a busy queue which contains a message for the node* $(i+2)$ *mod* 4.

queue $q_1$, to be sent to the node $(x_i + 2)$ *mod* $k$ on the shortest path.

- If $k = 2$, no message needs to move because it is already at its destination node.
- If $k = 3$, each message must travel just one link, because it may be consumed at the terminal node of that link. Each message is then directly sent to its consuming process and does not need to be stored in any queue $q_1$ of the next node. It follows that all messages can move.
- If $k = 4$, each message has two possibles paths. One solution is for instance to move the messages of the nodes 0 and 1 in $dir = 1$ while those of nodes 2 and 3 are moved in $dir = 0$ (see Figure 3). These movements are possible because in $dir = 1$ node 1 can move its message towards the queue of the outgoing link of the node 2 and in $dir = 0$ node 2 can also move its message toward the queue of the outgoing link of the node 1.
- In contrast to the previous cases, when $k \geq 5$, messages can not move. In fact, they must all move in the $dir = 1$ while the queue associated to each outgoing link in this direction is busy; this is evidently a blocked situation.

The only way to avoid such a blocking while routing on the shortest paths is the use of one *supplementary queue.* Associate to each outgoing link of any node, a supplementary queue $q_2$ which will be multiplexed with $q_1$. Figure 4 shows the new architecture of a node. However, even with the use of this supplementary

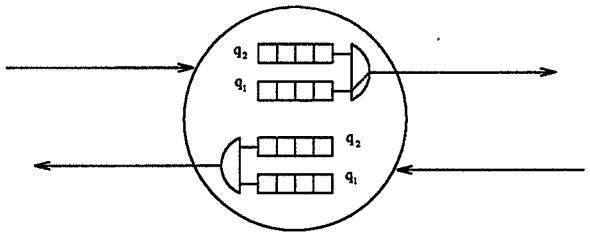


FIG. B.4 – *The new architecture of a node of a k-nodes ring: each outgoing link multiplexes two queues $q_1$ and $q_2$.*

queue, in the absence of some rules to use it, deadlock can occur again. Then, let $e$ be the end around connection of the $k$-nodes ring, namely the connection between the nodes of coordinates 0 and $k-1$.

- **Assertion** 1 : The following two rules allow deadlock free routing on the shortest paths.

  $R_1$ : At its source node a message is stored in the queue $q_1$ of its outgoing link,

  $R_2$ : A message originating from the queue $q_i$ of a node $x$ must be stored in the queue $q_i$ ($q_{i+1}$) of the next node $y$ if the outgoing link defined by the arc $(x, y)$ is not (is) a link of the connection $e$.

- **Proof:** The proof is straightforward. Indeed, now the node with coordinates $k-1$ can send its message stored in the queue $q_1$ to the queue $q_2$ of the node with coordinate 0 in direction 0. Node $j-1$ can then move its message in queue $q_1$ to node $j$ for $j = k-1$ down to 1. □

  Figure 5 illustrates the use of rule $R_2$. In fact rule $R_2$ means that to move through a link of $e$ a *penalty* of one new queue must be paid. So, in what follows, $q_2$ will be called a penalty queue. This reasoning remains correct when node $y$ is at the distance 2 from node $x$. Now suppose that node $x$ wants to send its message to any other node $y$ in the ring.

- **Assertion** 2 : Let $R_3$ be the following rule :

  route in the direction,

$$dir = \begin{cases} 1 & if\ ((y_i < x_i)\ and\ (d \geq k/2))\ or\ ((y_i > x_i)\ and\ (d \leq k/2)) \\ 0 & if\ ((y_i < x_i)\ and\ (d \leq k/2))\ or\ ((y_i > x_i)\ and\ (d \geq k/2)) \end{cases}$$

  $where\ d = |y_i - x_i|$

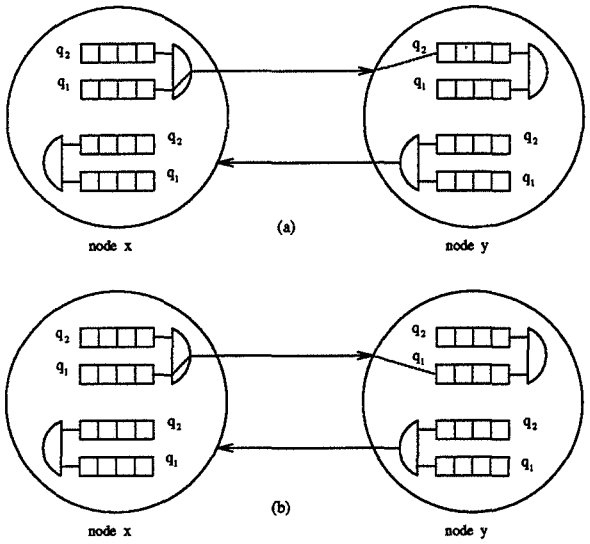


FIG. B.5 – *Rules for the use of the queues multiplexed on the same physical link. case (a) : The link $(x, y)$ belongs to the connection e and y is not the destination node. case (b) : The link $(x, y)$ does not belong to the connection e.*

Then rules $R_1$ and $R_2$ of assertion 1 and rule $R_3$ ensure a deadlock free routing on the shortest paths.

- **Proof:** Let us first prove that $R_3$ does ensure routing on the shortest paths. For a source node $x$ and a destination node $y$ let $d_j(x, y)$ be the length of the path from $x$ to $y$ in the direction $j$ :
  - if $(y_i < x_i)$ then :
    - $d_1(x, y) = y_i + k - x_i = k - (x_i - y_i) = k - |y_i - x_i|$
    - $d_0(x, y) = x_i - y_i = |y_i - x_i|$

    From where,

$$\begin{cases} d_1(x,y) \leq d_0(x,y) \iff |y_i - x_i| \geq k/2 \iff d \geq k/2, & \textit{and} \\ d_1(x,y) \geq d_0(x,y) \iff |y_i - x_i| \leq k/2 \iff d \leq k/2. \end{cases}$$

  - if $y_i > x_i$ then :
    - $d_1(x, y) = y_i - x_i = |y_i - x_i|$
    - $d_0(x, y) = x_i - y_i + k = k - (y_i - x_i) = k - |y_i - x_i|$

From where,

$$\begin{cases} d_1(x,y) \leq d_0(x,y) & \Longleftrightarrow & |y_i - x_i| \leq k - |y_i - x_i| \\ & \Longleftrightarrow & |y_i - x_i| \leq k/2 \iff d \leq k/2, \quad and \\ d_1(x,y) \geq d_0(x,y) & \Longleftrightarrow & |y_i - x_i| \geq k - |y_i - x_i| \\ & \Longleftrightarrow & |y_i - x_i| \geq k/2 \iff d \geq k/2. \end{cases}$$ □

We have shown that for a $k$-ary 1-cube one penalty queue is necessary and sufficient for deadlock free routing on the shortest paths. Next, we examine what arises for $n \geq 2$.

## B.3 Description of the algorithm

It should be mentioned that as a cartesian product of $n$ $k$-ary 1-cube, any node $x$ of a $k$-ary $n$-cube is in the intersection of $n$ $k$-ary 1-cube and can be located, in an orthogonal axis system of cardinality $n$, by $n$ coordinates $(x_i, 0 \leq i \leq n-1)$; $x_i$ is its coordinates in the $i$-th axis according to the $i$-th $k$-ary 1-cube. Then from the proof of Assertion 2, the length of a shortest path between two nodes $x$ and $y$ equals:

$$\sum_{i=0}^{n-1} Min(d_0(x_i, y_i), d_1(x_i, y_i)).$$

This distance can be performed in moving on the shortest path of the $k$-ary 1-cube of dimension $i$, in the decreasing order of $i$ such as to ensure each time $x_i = y_i$. This routing strategy is adapted from the $e$-cube routing widely used in the $k$-ary $n$-cube without wraparound connections, for which it is deadlock free [23] (see Figure 6). Meanwhile, using directly such a routing strategy can also result in deadlocks. As in the case of a $k$-ary 1-cube, to avoid deadlock while preserving routing on the shortest paths, it suffices to pay a penalty queue each time a message should travel through an end around connection. Because of the $e$-cube routing the maximum number of the end around links that a shortest path can contain does not exceed $n$. Assertion 2 can be applied again with $(n+1)$ queues. We have then proved the following result:

- **Assertion** 3: Given a $k$-ary $n$-cube with wraparound connections, with $(n+1)$ queues associated to each outgoing link $e$-cube routing under rules $R_1, R_2$ and $R_3$ does ensure deadlock free routing on shortest paths.

Let us now describe more formally the algorithm that a node $x$ has to execute to forward a message. This algorithm supposes that $x$ knows its neighbours nodes and then knows if an outgoing link is an end around one or not. The algorithm supposes also that a message $msg$ consists of a destination address $y$, a data to be transmitted and the number of the queue in which the message must be stored at

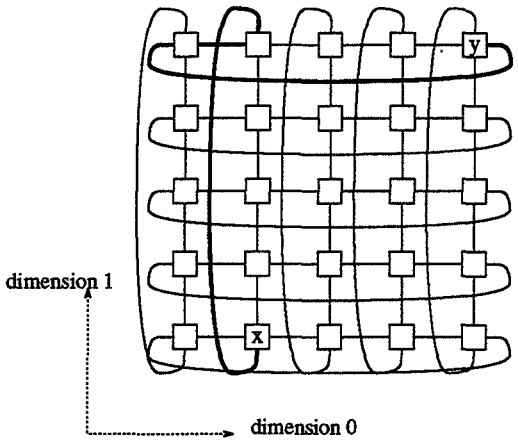


FIG. B.6 – *The shortest path between node x and y according to e-cube routing in a 5-ary 2-cube*

the next node. Let *dimension* and *direction* be respectively the dimension and the direction in which *msg* should be sent. The algorithm that node $x$ may execute according to Assertion 3 is as follows:

```
begin
    dimension ← Max{i ∈ E_k / x_i ≠ y_i};
    /* compute direction */
    d ← |y_dimension − x_dimension|;
    if (y_dimension < x_dimension) then
        if (d ≥ k/2) then
            direction ← 1
        else
            direction ← 0;
    else
        if (d ≥ k/2) then
            direction ← 0
        else
            direction ← 1;
    /* compute the address of the next node z */
    if direction = 1 then
        z_dimension ← ((x_dimension + 1) mod k)
    else
        z_dimension ← ((x_dimension − 1) mod k);
    z ← (x_0 ... x_dimension−1 z_dimension y_dimension+1 ... y_k−1);
```

if ($msg$ has been generated at node $x$) then
$current\_queue\_number \leftarrow 1$;
if ($(x, z)$ is an end around link) then
$next\_queue\_number \leftarrow current\_queue\_number + 1$
else
$next\_queue\_number \leftarrow current\_queue\_number$;
send $msg$ from ($q_{current\_queue\_number}$ of $x$) on $(x, z)$ to ($q_{next\_queue\_number}$ of $z$);
end

## B.4 Analysis of the algorithm

The algorithm described in section 3 verifies two of the main criteria for designing a routing algorithm : deadlock avoidance and routing on shortest paths. The price to pay for making compatible these two criteria is a penalty expressed in terms of the number of queues to be associated to each outgoing link of the network. For a $k$-ary $n$-cube with wraparound connections, at least $(n+1)$ queues are required.

The routing on the shortest paths is obviously due to the use of the $e$-cube routing algorithm. For each dimension, the routing direction is determined by the shortest path between the projection on that dimension of the source node and the destination node. The deadlock avoidance property comes both from the $e$-cube routing algorithm and the ordering of the penalty queues of two adjacent nodes. When the end around connections are not used, $e$-cube routing ensures deadlock avoidance. The messages travel from the queue $q_1$ of the current node to the queue $q_1$ of the next node until they attain their destinations. No penalty queue is required. When the routing paths contain at least one end around connection the deadlock free property is ensured by both $e$-cube routing and the ordering of the queues according to the rule $R_2$.

On other hand, the proposed algorithm needs a constant memory space for representing the routing function as it is an extended version of the $e$-cube routing algorithm. Thus, it is well adapted to VLSI implementation mainly for small values of $n$ as it is the case in practice.

The unique shortest paths that the algorithm allow while guaranteeing deadlock avoidance are the ones defined by $e$-cube routing. However other kinds of shortest paths can also be considered. To that purpose, the principle of the penalty queue has to be extended. Let us consider a longest shortest path of a $k$-ary $n$-cube; its length equals $n\lfloor k/2 \rfloor$. One can prove easily that $(n\lfloor k/2 \rfloor - 1)$ queues per outgoing links are sufficient for deadlock free routing on any shortest path. Indeed consider any node $x$ of the network, the unique type of switchings allowed

by the algorithm between an incoming link of $x$ and an outgoing one, are the switchings from an incoming link in a dimension $\Delta_{in}$ to an outgoing link in a dimension $\Delta_{out}$ such as $\Delta_{out} \leq \Delta_{in}$. Thus, for any path, the maximum number that such switching may be infringed can not exceed its length minus 1. Therefore, adding a new penalty queue each time the switching rule is infringed, results in $(n\lfloor k/2 \rfloor - 1)$ queues per outgoing link for the longest shortest paths.

This bound is too rough. Indeed a path which realises $(n\lfloor k/2 \rfloor - 1)$ penalties, corresponds to a strictly decreasing sequence of integers that represents the dimensions. But with $n$ dimensions the length of such a sequence does not exceed $n$. It follows that less than $(n\lfloor k/2 \rfloor - 1)$ queues per outgoing link allow deadlock free routing on all the shortest paths. Fully adaptive and fault tolerant routing can then be envisaged with less memory space than the $2^{n-2}(n+1)$ queues per outgoing link stated by LINDER and HARDEN in [69].

It should be finally mentionned that more queues allows to relax the constraint on the length of the paths, the algorithm then allow more paths between any couple of $(source, destination)$ nodes; more adaptive and more fault tolerant routing may then be considered without disturbing the deadlock property.

## B.5 Conclusion

This paper reports a routing algorithm for the $k$-ary $n$-cubes with wraparound connections. The algorithm is deadlock free and allows to route on shortest paths. $(n+1)$ queues per outgoing communication link are required to ensure these two properties. This makes the algorithm well appropriated for high radix (high $k$) $k$-ary $n$-cube.

The algorithm is an extended version for the torus of the $e$-cube routing algorithm widely used for meshs. It is then well-appropriate for VLSI implementation with lower value of $n$.

To allow deadlock free routing on all the shortest paths, less than $(n\lfloor k/2 \rfloor - 1)$ queues per outgoing link are required. That allows to envisage adaptive even fault tolerant routing on shortest paths. Relaxing the constraint on the length of the paths generates more paths between the couples of (source,destination) nodes; allowing more adaptivity and fault tolerant routing.

The price to pay for taking into account more routing criteria is more queues per outgoing link and more complex control logic for route selection than does the algorithm described in section 3.

To be meaningful, the induced algorithm should be simulated from realistic workload distribution.

# Bibliographie


[1] DOC 72 TDS 225 00. *ANSI C toolset, reference manual, user manual.* INMOS Limited, August, 1990.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, 1974.

[3] J. Annot and R. Van Twist. A novel deadlock free and starvation free packet switching communication processor. *LNCS 258*, (68-85), June 1987.

[4] E.A. Aschroft and W.W. Wadge. *Lucid, the Dataflow Programming Language.* Academic press, 1985.

[5] B. Awerbuch, A. Bar-Noy, N. Linial, and D. peleg. Improved routing strategies with succinct tabless. *Journal of Algorithms*, (11), 1990.

[6] E. Bakker, J. Van Leeuwen, and R. Tan. Prefix routing schemes in dynamic networks. Ruu-cs-90-10, Dept of Computer Science, Utrecht University, March 1990.

[7] H. Bal, J. Steiner, and A. Tanenbaum. Programming languages for distributed computing systems. *ACM Computing Surveys*, 21(3), September 1989.

[8] C. Berge. *Théorie des Graphes.* Gauthier-Villars, 1983.

[9] D. Blough and A. Pelc. Diagnosis and repair in multiprocessor systems. *IEEE Transactions on Computers*, 42(2), February 1993.

[10] A. Bouabdallah and M.B. Hadim. A distributed algorithm for finding an Eulerian cycle in networks. *In IASTED International Conference Euro-PDS'97*, Barcelone, Espagne, June 9-11, 1997.

[11] A. Bouloutas and P. Gopal. Some graph partitionning problems and algorithms related to routing in large computer networks. In *The 9-th Conference on Distributed Computing*, June 1989.

[12] J. Bruck, R. Cypher, and C. Ho. Fault-tolerant meshes and hypercubes with minimal numbers of spares. *IEEE Transactions on Computers*, 42(9), September 1993.

[13] H. Buhrman, J-H. Hoepman, and P. Vitányi. Optimal routing tables. *15th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, ACM PRESS.

[14] C. Chan and T. Yum. An algorithm for detecting and resolving store and forward deadlocks in packet-switching networks. *IEEE Transactions on Communications*, 35(8), 1987.

[15] A. Chien and J. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. In 268-277, editor, *Proceedings of the Int Symp on Comp Architecture*, May-1992.

[16] W. Chou, J. Powell, and A. Bragg. Comparative evaluation of deterministic and adaptive routing, flow control in computer networks. *IFIP, North-Holland Publishing Company*, 1979.

[17] M. Cosnard and F. Desprez. Quelques architectures de nouvelles machines. *Calculateurs Parallèles*, 21:29–58, mars 1994.

[18] Inc Cray Research. Cray T3D software overview technical note. *Special Draft Edition for IEEE Supercomputing'92*, 1992.

[19] J. Crowcroft and Z. Wang. Analysis of shortest-path routing algorithms in a dynamic network environment. *Computer Communication Review*, 22, 1992.

[20] R. Cypher and L. Gravano. Requirements for deadlock-free, adaptive packet routing. *SIAM Journal on Computing*, (23(6)), December 1994.

[21] R. Cypher and L. Gravano. Storage-efficient, deadlock-free packet routing algorithms for torus networks. *IEEE Transactions on Computers*, 43(12), 1994.

[22] J. Dally. Performance analysis of $k$-ary $n$-cube interconnection networks. *IEEE Transactions on Computers*, 39(6), 1990.

[23] W. Dally and C. Seitz. The torus routing chip. *Distributed Computing*, 1, 1986.

[24] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, cX-36(5), 1987.

[25] J. de Rumeur. *Communications dans les réseaux de processeurs.* Masson, 1994.

[26] M. Debbage, M. Hill, and D. Nicole. Global communications on locally connected message-passing parallel computers. *Concurrency: Practice and Experience*, 5(6), September 1993.

[27] R. Despons. Conception d'une Machine Virtuelle pour les Systèmes Parallèles à Diffusion. Thèse, INPG, 1996.

[28] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12), December 1993.

[29] S. Dutt and J. Hayes. Some practical issues in the design of fault-tolerant multiprocessors. *IEEE Transactions on Computers*, 41(5), May 1992.

[30] D. Etiemble. *Architectures des Processeurs* RISC. Armand Colin, 1991.

[31] S. Felperin, L. Gravano, G. Pifarré, and J. Sanz. Routing techniques for massively parallel communication. In *Proceedings of the IEEE*, volume 79-4, 1991.

[32] M. Flynn. Some computer organization and their effectiveness. *IEEE Transactions on Computers*, C-21, 1972.

[33] P. Fraigniaud and C. Gavoille. Memory requirement for universal routing schemes. *14th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, ACM PRESS, 223-230.

[34] P. Gambosi, D. Bovet, and D. Menasce. A detection and removal of deadlocks in store and forward communication networks. *Performance of Computer-Communication Systems, RUDIN H. and BUX W. (ed), IFIP*, 1984.

[35] M.R. Garey and D S. Jonhson. *Computers and Intractability: A Guide to the Theory of NP-Completness.* Freeman, 1979.

[36] P. Gaughan and S. Yalamanchili. Adaptive routing protocols for hypercube interconnection networks. *IEEE, Computers*, 26(5), May 1993.

[37] B. Gavish, P. Merlin, and P. Schweitzer. Minimal buffer requirements for avoiding store-and- forward deadlock. Technical Report 6672, IBM Thomas J Watson Research Center, 1977.

[38] C. Gavoille. Complexité Mémoire du Routage dans les Réseaux Distribués. Thèse de doctorat, ENS-LYON, 1996.

[39] D. Gelernter. A dag-based algorithm for prevention of store-and-forward deadlock in packet networks. *IEEE Transactions on Computers*, C-30(10), 1981.

[40] M. Gerla and L. Kleinrock. Flow control: A comparative survey. *IEEE Transactions on Communications*, 28, 1980.

[41] C. Germain-Renaud. Etude des Mécanismes de Communication pour une Machine Massivement Parallèle: MEGA. Thèse de Doctorat, Université Paris-Sud, Centre d'Orsay, LRI, 1992.

[42] C. Germain-Renaud and J. Sansonnet. *Les Ordinateurs Massivement Parallèles.* Armand Colin, 1991.

[43] A. Giessler, J. Hanle, A. Koning, and A. Pade. Free buffer allocation-an investigation by simulation. *Computer Networks*, (2), 1978.

[44] C. Glass and L. Ni. The turn model for adaptive routing. *ACM*, Com-28(No 4), July 1992.

[45] I. Gopal. Prevention of store-and-forward deadlock in computer networks. *IEEE Transactions on Communications*, COM-33(12), 1985.

[46] K. Gunther. Prevention of deadlocks in packet-switched data transport systems. *IEEE Transactions on Communications*, 29(4), 1981.

[47] B. Hadim and I. Sakho. Dérivation de stratégies de routage sans interblocage dans les réseaux de processeurs. *Actes de RenPar'8, Bordeaux, 20-24 Mai*, 1996.

[48] B. Hadim and I. Sakho. Du routage des messages dans les architectures parallèles à mémoire distribuée. *TSI vol 16 n°3*, 1997.

[49] B. Hadim and I. Sakho. Une méthode de régulation du rapport correction/efficacité d'un schéma de communication. *Actes de RenPar'7, Lyon-France*, Mai-Juin 1995.

[50] M.B. Hadim and I. Sakho. The multi-level communication: Minimal, deadlock free, storage optimal routing for torus networks. *Soumis à Journal of Parallel and Distributed Computing.*

[51] M.B. Hadim and I. Sakho. A new methodology for deriving deadlock-free routing strategies in processor networks. *Accepté à International Conference on Parallel and Distributed Computing Systems*, Louisiane USA, Oct 1997.

[52] M.B. Hadim and I. Sakho. Minimal, deadlock free and $o(n)$ space memory routing for $k$-ary $n$-cubes with wraparound connections. *International Conference on Parallel and Distributed Processing, Techniques and Applications*, Athens, Georgia(USA), 1995.

[53] P. Hilbers and J. Lukkien. Deadlock-free message routing in processor networks. *Distributed Computing*, 3:178-186, 1989.

[54] C.D. Hillis. The connection machine. *Scient Amer*, 256(6), June 1987.

[55] H. Hofestadt, A. Klein, and E. Reyzl. Performance benefits from locally adaptive interval routing in dynamically switched interconnection networks. In A.Bode, editor, *Proceedings of the EDMCC2, LNCS 487 Springer Verlag*, 1991.

[56] INMOS. *The T9000 Transputer Products Overview Manual.* SGS-THOMSON, 1991.

[57] C. Jesshope, P. Miller, and J. Yantchev. High performance communication in processor networks. *ACM Computer Architecture News*, 17(7), 1989.

[58] S. Johnsson and C. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transaction on Computers*, 38(9):1249-1268, 1989.

[59] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3, 1979.

[60] P. Kermani and L. Kleinrock. A tradeoff study of switching systems in computer communication networks. *IEEE Transactions on Computers*, 29(12), 1980.

[61] R. Keryell. POMP : d'un Petit Ordinateur Massivement Parallel SIMD à base de Processeurs RISC - Concepts, Étude et Réalisation . Thèse, Université Paris XI Orsay, 1992.

[62] J. Kim and K. Shin. Deadlock-free fault-tolerant routing in injured hypercubes. *IEEE Transactions on Computers*, 42(9), 1993.

[63] L. Kleinrock and F. Kamoun. Hierarchical routing for large networks: Performance evaluation and optimization. *Computer Networks*, 1, 1977.

[64] J. Van Leeuwen and R. Tan. Interval routing. *The Computer Journal*, 30(4), 1987.

[65] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes.* Morgan Kaufmann, 1992.

[66] F.T. Leighton, B.M. Maggs, A.G. Ranade, and S.B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17, 1994.

[67] Q. Li. Minimum deadlock-free message routing restrictions in binary hypercubes. *Parallel and Distributed Computing*, 15, 1992.

[68] X. Lin and L. Ni. Deadlock-free multicast wormhole routing in multicomputer networks. *CACM*, 394, September 1991.

[69] D. Linder and J. Harden. An adaptive and fault tolerant wormhole routing strategy for *k-ary n-cubes. IEEE Transactions on Computers*, 40(1), January 1991.

[70] P. Michallon. Schémas de Communications Globales dans les Reseaux de Processeurs: Application à la Grille Torique. Thèse, INPG, 94.

[71] S. Miguet. Programmation Dynamique et Traitement d'Images sur Machines Parallèles à Mémoire Distribuée. Thèse ENS-Lyon, 1990.

[72] L. Mugwaneza. Contrôle de la Communication dans les Machines Parallèles à Mémoire Distribuée: Contribution au Routage Automatique des Messages. Thèse, INPG, 1993.

[73] L. Mugwaneza, T. Muntean, and I. Sakho. A deadlock free routing algorithm with network size independent buffering space. *Proceedings of CONPAR90-VAPP IV, Zurich Switterland*, September 1990.

[74] J. Ngai and C. Seitz. A framework for adaptive routing. Technical Report 5246, Computer Science Departement, California Institute of Technology, 1987.

[75] L.M. Ni and K. Panda. A report of the icpp '94 panel on *sea of interconnection networks: what's your choice?* 1994.

[76] M. Ni and K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26(2), 1993.

[77] M. Norman, N. Radcliffe, and L. Clarke. What price regularity? *Concurrency: Practice and Experience*, 2(1), March 1990.

[78] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3), July 1989.

[79] F.S. Roberts. *Applied Combinatorics.* Prentice-Hall Inc, 1984.

[80] A. Roscoe. Routing messages through networks : an exercise in deadlock avoidance. *Workshop on Parallel Programming of Transputers Based Machines*, Grenoble Sept 14-16, 1987.

[81] Y. Saad and M. Schultz. Topological properties of hypercubes. *IEEE Transaction on Computers*, C-37:867-872, 1988.

[82] I. Sakho, L. Mugwaneza, and Y. Langue. Routing with compact routing tables : Interval labelling scheme for generalized meshes. *Proceedings of the IFIP WG10.3 Working Conference on Applications in Parallel and Distributed Computing*, Caracas 1994.

[83] J.P. Sansonnet. *Concepts d'Architectures Avancées.* LRI - Bât 490 Université de Paris XI - Orsay, 1990-1991.

[84] N. Santoro and R. Khatib. Labelling and implicit routing in networks. *The Computer Journal*, 28(1), 1985.

[85] V. Sarkar. *Partitionning and Scheduling Parallel Programs for Multiprocessors.* Research monographs in parallel and distributed computing, The MIT press, Cambridge Massachusset, 1989.

[86] M. Schwartz and T. Stern. Routing techniques used in computer communication networks. *IEEE Transactions on Communication*, Com-28(No 4), April 1980.

[87] M. Syska. Communication dans les Architectures à Mémoire Distribuée. Thèse, Université de Nice-Sophia Antipolis, 1992.

[88] A. Tanenbaum. *RESEAUX, Architectures, Protocoles, Applications.* Inter-Editions, 1990.

[89] G. Tel. *Introduction to Distributed Algorithms.* Cambridge Universiy Press, 1994.

[90] Y.B. Langue Tsobgny. PARX: Architecture de Noyau de Système d'Exploitation Parallèle. Thèse INPG, 1991.

[91] N. Gonzalez Valenzuela. PARX: Noyau de Système pour les Ordinateurs Massivement Parallèles. Contrôle de la Communication entre Processus. Thèse INPG, 1991.

[92] L. Valliant. A scheme for fast parallel communication. *SIAM, J.COMPUT*, 11-2 May, 1982.

[93] L. Valliant. General purpose parallel architecture. Tr-07-89, Harvard University, 1989.

[94] P. Waille. Architectures Parallèles à Connectique Programmable, Reconfiguration et Routage. Thèse, INPG, 1991.

# Table des figures

# Résumé

Dans les architectures parallèles à mémoire distribuée, la communication entre processus est un des facteurs de performance les plus importants pour les applications. Le système qui en a la charge, i.e, *le noyau de communication*, doit intégrer une fonctionnalité essentielle pour de telles architectures : *le routage des messages*. Cette fonctionnalité est assurée par une composante spécifique du noyau de communication : *le noyau de routage*, dont le rôle est l'acheminement d'un message d'un nœud émetteur vers un nœud récepteur.

L'acheminement des messages nécessite une *stratégie de routage* qui spécifie les chemins de communication pour toute paire de processeurs (source,destination) du réseau d'interconnexion. Une telle stratégie de routage doit satisfaire d'une part, des *critères de correction* et d'autres part, des *critères d'efficacité*.

Le but de cette thèse est la conception de stratégies de routage pour les réseaux de processeurs qui satisfont à la fois, les critères de correction et les critères d'efficacité. Nous proposons une méthode de conception de stratégies de routage, permettant par une démarche incrémentale, de satisfaire les deux types de critère : *la communication multi-niveaux* et le *schéma de communication primaire* associé.

Pour mesurer l'efficacité de la méthode, nous l'appliquons à un réseau particulier : *la grille torique*. Les différents algorithmes de routage obtenus sont corrects et très efficaces.

Nous proposons également une technique d'implantation de notre méthode de routage, permettant le calcul des tables de routage directement sur le réseau de processeurs. Cette technique permet ainsi l'obtention d'un système *auto-constructif*.

**Mots clés** : Architectures parallèles, Communication, Routage des messages, Communication multi-niveaux, Schéma de communication primaire, Grille torique.